Dashboard manual

Introduction

This document serves as a guide for the dashboard, in particular the frontend part of the codebase. It describes how to run and test the frontend, what functionality the dashboard contains and how to expand the code to include more visualization types.

Frontend setup

A number of steps need to be taken to get the dashboard itself running for the first time:

1. install npm (https://nodejs.org/en/download/) by picking the version for your operating system
2. run the installer using the default settings
3. if you do not use a compatible IDE already and wish to continue developing the software, download and open vscode (Download Visual Studio Code - Mac, Linux, Windows)
4. create a folder and clone the repository there, e.g. through a command line
5. open a terminal or powershell in the folder (or through an integrated terminal in your IDE) and navigate to the medctrl-frontend folder
6. run “npm install” in the terminal to install all the dependencies
7. if you want to run the frontend without locally running the backend, add a .env.local file in the medctrl-frontend folder with the line “REACT_APP_RDR_DEV=true”, this tells the program to use the online development server instead
8. run “npm start” to get the application running, you can now visit a local version at localhost:3000

After the first time you will only need to run “npm install” when new dependencies have been added. Otherwise you just need to run “npm start”. You will always need to do this from a terminal pointing to the medctrl-frontend folder.

Code styling tools

We use Prettier to reformat the code and the linter ESLint to find problematic code patterns. You can apply Prettier by running “npx prettier --write ./” in a terminal in the medctrl-frontend folder or a folder within the medctrl-frontend folder to run the tool over all files in the folder. This will automatically change ill formatted code to make it correspond to a default style. You can run “npx eslint ./” in the medctrl-frontend folder or a folder within the medctrl-frontend folder to run ESLint over all files in the folder. This will give warnings about potentially bad code style.

Testing tools

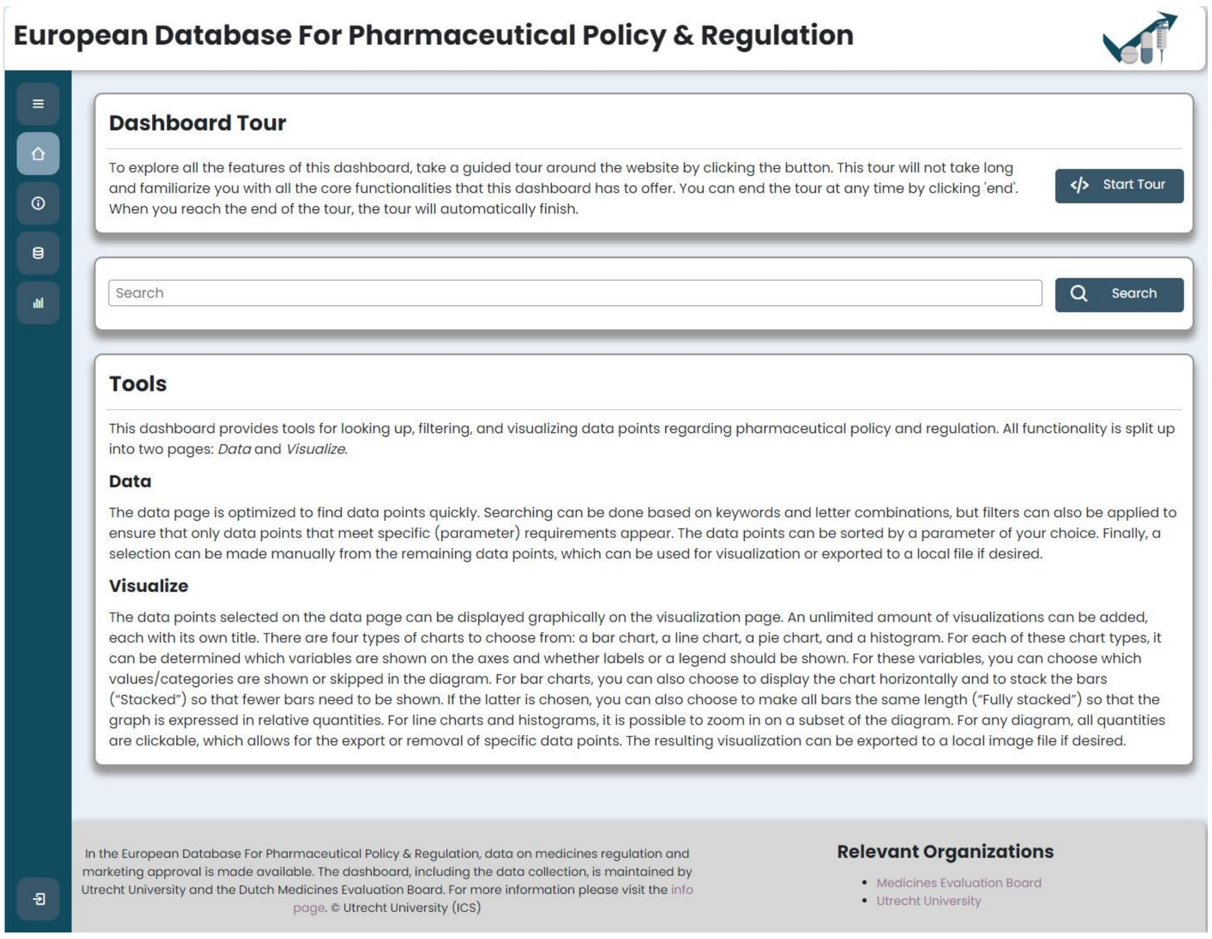
We have included various tests for our codebase. Unit tests were written using Jest and dom-testing-library. You can add a test in a __test__ folder by creating “[file].test.js”. You can run a test by “npm run test”. These folders must be within the medctrl-frontend folder. You can add the filename at the end of this command to only run the tests in that file. You can run “npm test -- --watchAll --coverage” to run all tests and keep track of which lines they have covered. This also creates an html document, see *medctrl-frontend/coverage/lcov-report/index.html*. This document will be rewritten after every run. Viewing this document in your browser will provide you with a more readable version of the coverage report. Integration and system tests were written using Selenium with Python, a separate manual has been included for this in the repository.

Frontend functionality

We will briefly describe the current functionality of the dashboard here. An extensive walkthrough of the system can already be found through the interactive tour functionality on the home page.

Home page

When you enter the website, you first see the home page. The home page contains a guided tour through the website, some general information on how to use the website and a search bar. When you use the search bar to search for a term, you are redirected to the data page, where the search results will be shown.



European Database For Pharmaceutical Policy & Regulation

Dashboard Tour

To explore all the features of this dashboard, take a guided tour around the website by clicking the button. This tour will not take long and familiarize you with all the core functionalities that this dashboard has to offer. You can end the tour at any time by clicking 'end'. When you reach the end of the tour, the tour will automatically finish.

Tools

This dashboard provides tools for looking up, filtering, and visualizing data points regarding pharmaceutical policy and regulation. All functionality is split up into two pages: *Data* and *Visualize*.

Data

The data page is optimized to find data points quickly. Searching can be done based on keywords and letter combinations, but filters can also be applied to ensure that only data points that meet specific (parameter) requirements appear. The data points can be sorted by a parameter of your choice. Finally, a selection can be made manually from the remaining data points, which can be used for visualization or exported to a local file if desired.

Visualize

The data points selected on the data page can be displayed graphically on the visualization page. An unlimited amount of visualizations can be added, each with its own title. There are four types of charts to choose from: a bar chart, a line chart, a pie chart, and a histogram. For each of these chart types, it can be determined which variables are shown on the axes and whether labels or a legend should be shown. For these variables, you can choose which values/categories are shown or skipped in the diagram. For bar charts, you can also choose to display the chart horizontally and to stack the bars (“Stacked”) so that fewer bars need to be shown. If the latter is chosen, you can also choose to make all bars the same length (“Fully stacked”) so that the graph is expressed in relative quantities. For line charts and histograms, it is possible to zoom in on a subset of the diagram. For any diagram, all quantities are clickable, which allows for the export or removal of specific data points. The resulting visualization can be exported to a local image file if desired.

In the European Database For Pharmaceutical Policy & Regulation, data on medicines regulation and marketing approval is made available. The dashboard, including the data collection, is maintained by Utrecht University and the Dutch Medicines Evaluation Board. For more information please visit the info page. © Utrecht University (ICS)

Relevant Organizations

- Medicines Evaluation Board
- Utrecht University

Info page

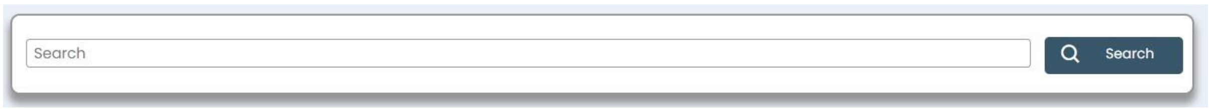
The info page describes the purpose of the dashboard, and the organizations in charge of, namely Utrecht University and the Medicine Evaluation Board (CBG). It also contains some information on other affiliated parties.

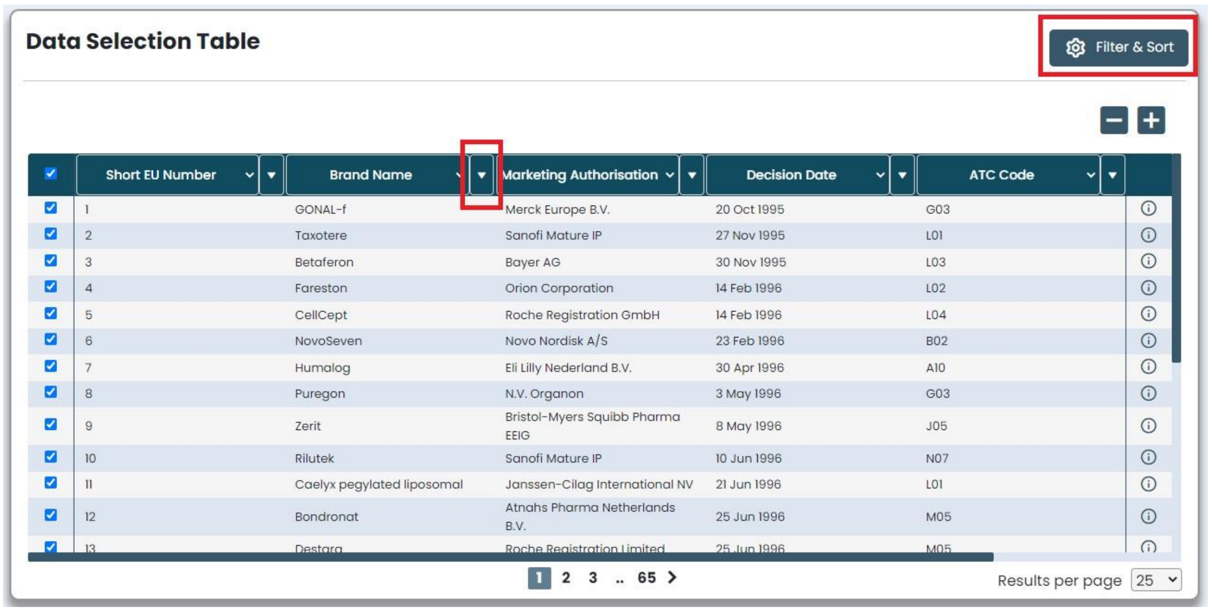
Data page

The data page consists of three parts: the search bar, the upper table that contains all the medicine data and the lower table that shows which data is currently selected.

Filtering and sorting data

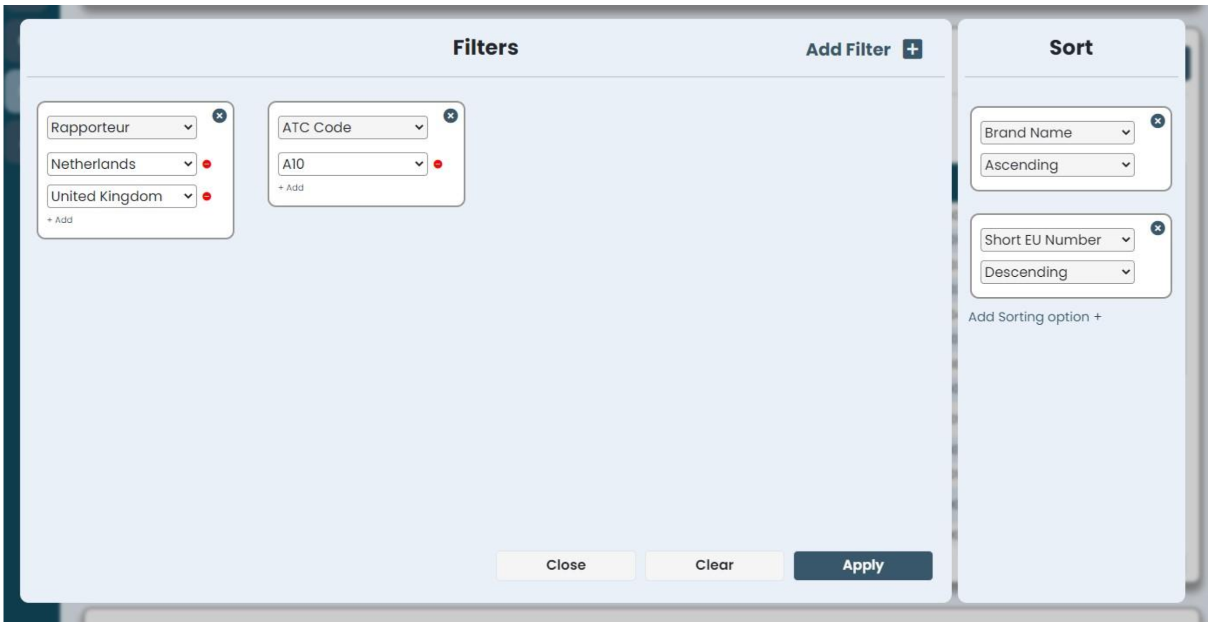
The search bar can be used to quickly search for medicines matching the search terms. The results are shown in the upper table. You can also use the filter menu to filter the data in this table. You can specify which value(s) a chosen variable can have, you can also choose a range when dealing with numerical data. You can add multiple filters. A multilevel "first by then by" sorting order can be specified in the sorting menu. Sorting on only one attribute can also be done by clicking the arrow button in the corresponding column header. Repeatedly clicking the in-column sorting button will alternate the sorting order between ascending and descending order. Non-available “NA” values are always placed last, irrespective of sorting order.




Data Selection Table

✓	Short EU Number	Brand Name	Marketing Authorisation	Decision Date	ATC Code
✓	1	GONAL-f	Merck Europe B.V.	20 Oct 1995	G03
✓	2	Taxotere	Sanofi Mature IP	27 Nov 1995	L01
✓	3	Betaferon	Bayer AG	30 Nov 1995	L03
✓	4	Fareston	Orion Corporation	14 Feb 1996	L02
✓	5	CellCept	Roche Registration GmbH	14 Feb 1996	L04
✓	6	NovoSeven	Novo Nordisk A/S	23 Feb 1996	B02
✓	7	Humalog	Eli Lilly Nederland B.V.	30 Apr 1996	A10
✓	8	Puregon	N.V. Organon	3 May 1996	G03
✓	9	Zerit	Bristol-Myers Squibb Pharma EEIG	8 May 1996	J05
✓	10	Rilutek	Sanofi Mature IP	10 Jun 1996	N07
✓	11	Caelyx pegylated liposomal	Janssen-Cilag International NV	21 Jun 1996	L01
✓	12	Bondronat	Atnahs Pharma Netherlands B.V.	25 Jun 1996	M05
✓	13	Destara	Roche Registration Limited	25 Jun 1996	M05

1 2 3 .. 65 › Results per page 25

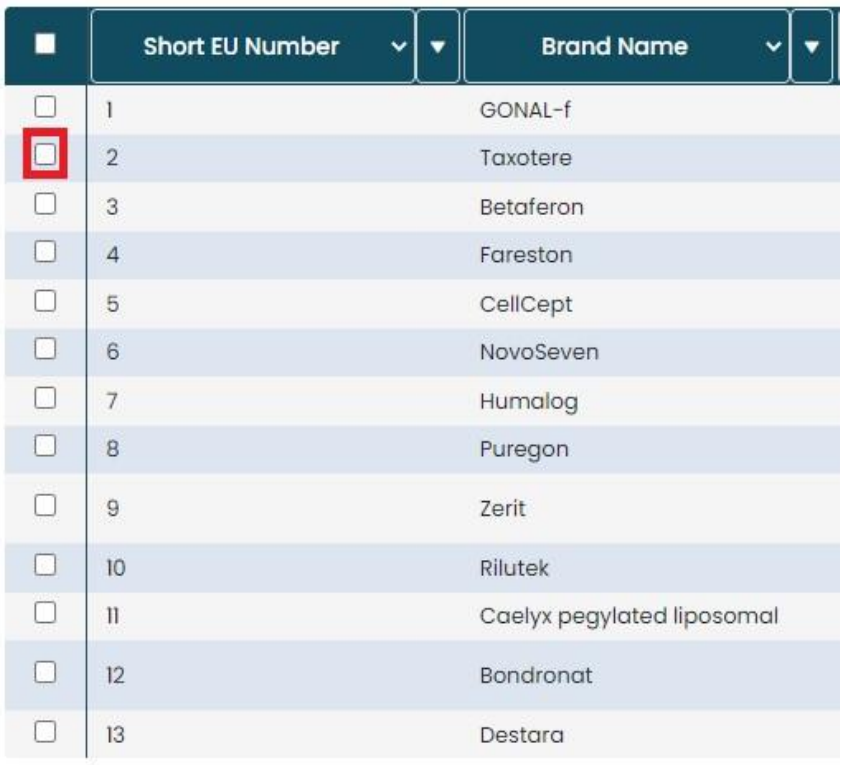


Filters: Rapporteur — Netherlands, United Kingdom; ATC Code — A10

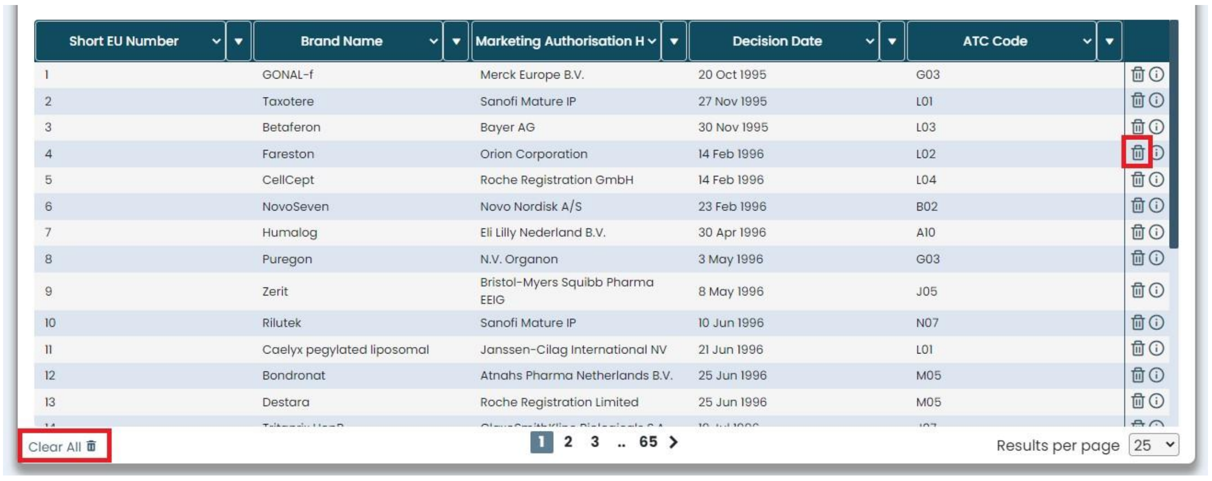
Sort: Brand Name — Ascending; Short EU Number — Descending

Selecting/deselecting data

Data entries can be added to your selected data entries in the lower table by checking their checkboxes in the upper table. Selected entries can be removed from your selection by unchecking their checkbox in the upper table or by pressing the garbage bin icon in the lower table. All data entries can be simultaneously removed from the lower table by pressing the ‘Clear all’ sign at the left side underneath the lower table.



☐	Short EU Number	Brand Name
☐	1	GONAL-f
☐	2	Taxotere
☐	3	Betaferon
☐	4	Fareston
☐	5	CellCept
☐	6	NovoSeven
☐	7	Humalog
☐	8	Puregon
☐	9	Zerit
☐	10	Rilutek
☐	11	Caelyx pegylated liposomal
☐	12	Bondronat
☐	13	Destara

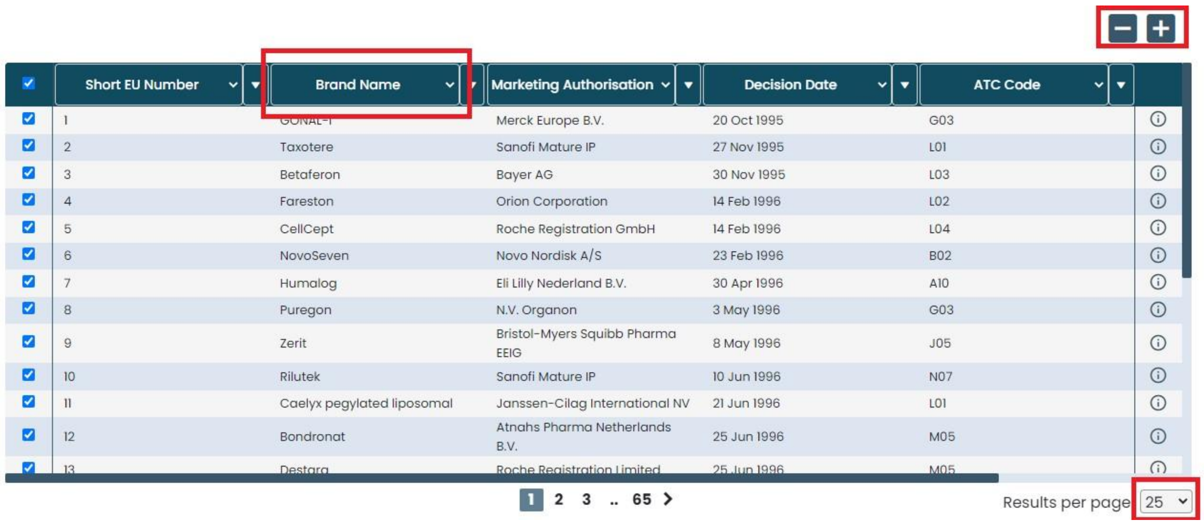


Short EU Number	Brand Name	Marketing Authorisation H	Decision Date	ATC Code
1	GONAL-f	Merck Europe B.V.	20 Oct 1995	G03
2	Taxotere	Sanofi Mature IP	27 Nov 1995	L01
3	Betaferon	Bayer AG	30 Nov 1995	L03
4	Fareston	Orion Corporation	14 Feb 1996	L02
5	CellCept	Roche Registration GmbH	14 Feb 1996	L04
6	NovoSeven	Novo Nordisk A/S	23 Feb 1996	B02
7	Humalog	Eli Lilly Nederland B.V.	30 Apr 1996	A10
8	Puregon	N.V. Organon	3 May 1996	G03
9	Zerit	Bristol-Myers Squibb Pharma EEIG	8 May 1996	J05
10	Rilutek	Sanofi Mature IP	10 Jun 1996	N07
11	Caelyx pegylated liposomal	Janssen-Cilag International NV	21 Jun 1996	L01
12	Bondronat	Atnahs Pharma Netherlands B.V.	25 Jun 1996	M05
13	Destara	Roche Registration Limited	25 Jun 1996	M05

Clear All 1 2 3 .. 65 Results per page 25

Changing layout

The columns can be ‘swapped’ by changing which column is shown where by selecting a different variable in the header. Note that just changing to a different variable does not mean that the column of the other variable is also changed, resulting in a duplication rather than a swap. The plus and minus signs can be used to add, respectively remove columns. There will always be at least 5 columns. The amount of data entries visible can be changed by multiples of 25, up to 300 per page.

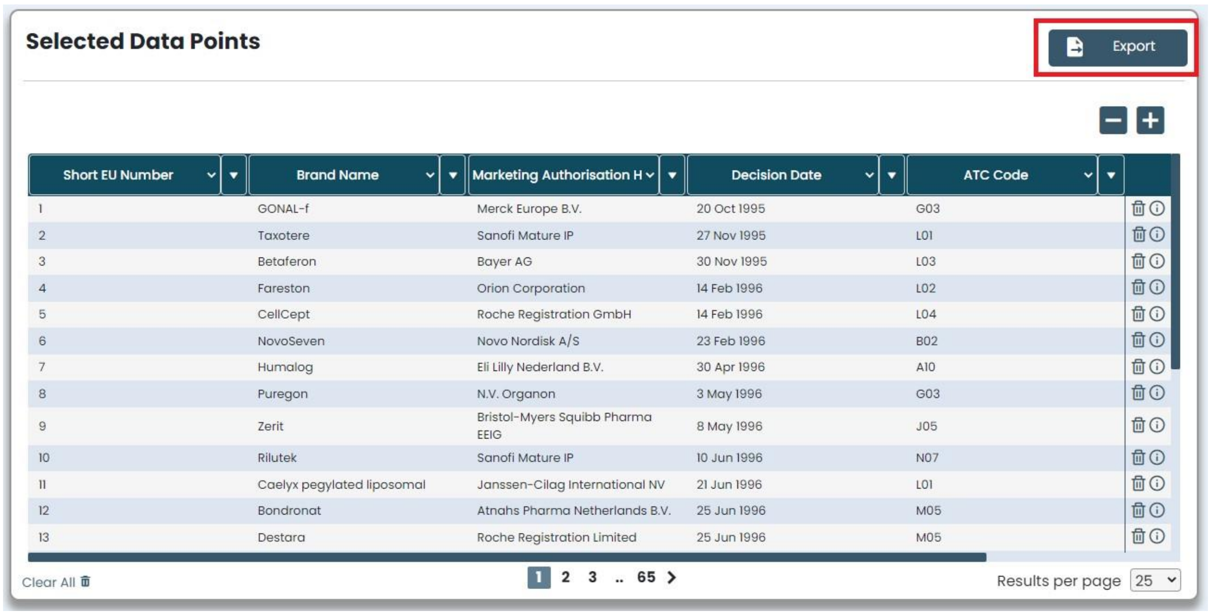


✓	Short EU Number	Brand Name	Marketing Authorisation	Decision Date	ATC Code
✓	1	GONAL-f	Merck Europe B.V.	20 Oct 1995	G03
✓	2	Taxotere	Sanofi Mature IP	27 Nov 1995	L01
✓	3	Betaferon	Bayer AG	30 Nov 1995	L03
✓	4	Fareston	Orion Corporation	14 Feb 1996	L02
✓	5	CellCept	Roche Registration GmbH	14 Feb 1996	L04
✓	6	NovoSeven	Novo Nordisk A/S	23 Feb 1996	B02
✓	7	Humalog	Eli Lilly Nederland B.V.	30 Apr 1996	A10
✓	8	Puregon	N.V. Organon	3 May 1996	G03
✓	9	Zerit	Bristol-Myers Squibb Pharma EEIG	8 May 1996	J05
✓	10	Rilutek	Sanofi Mature IP	10 Jun 1996	N07
✓	11	Caelyx pegylated liposomal	Janssen-Cilag International NV	21 Jun 1996	L01
✓	12	Bondronat	Atnahs Pharma Netherlands B.V.	25 Jun 1996	M05
✓	13	Destara	Roche Registration Limited	25 Jun 1996	M05

1 2 3 .. 65 Results per page 25

Exporting data

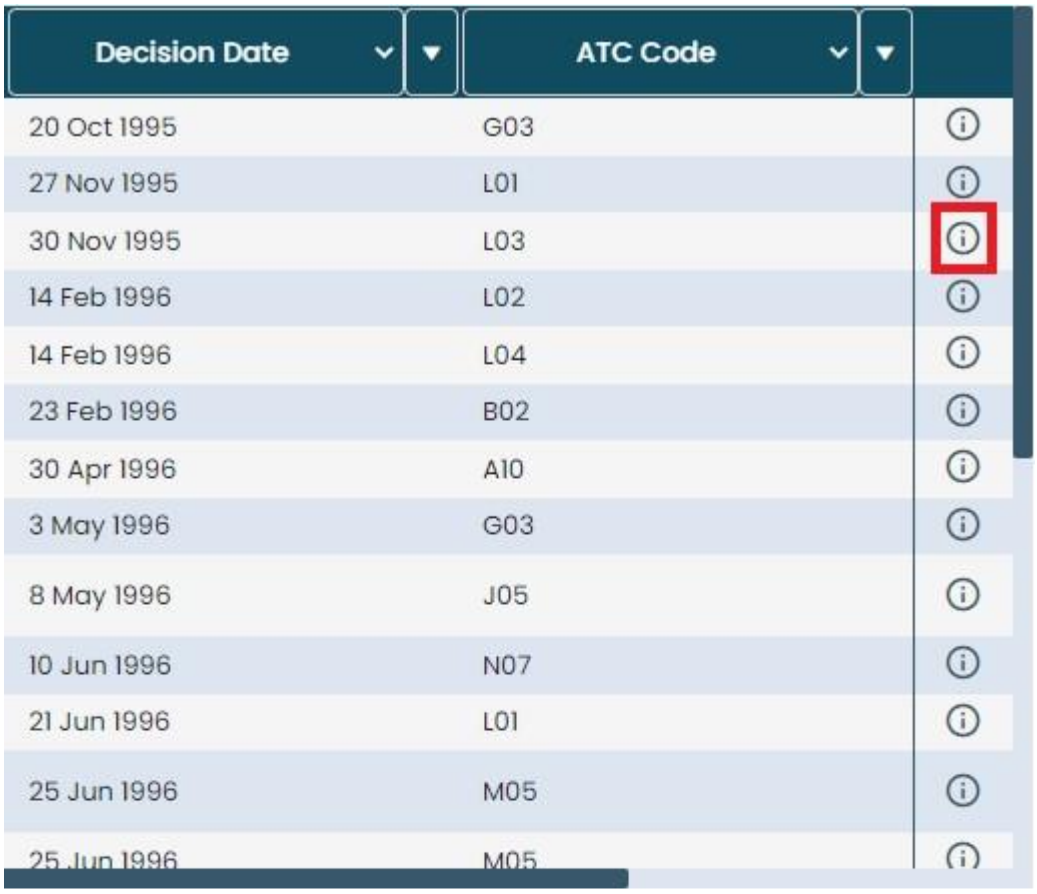
All selected data can be downloaded simultaneously by clicking the export button in the lower table. Various export types can be chosen, including a custom separator.


Selected Data Points

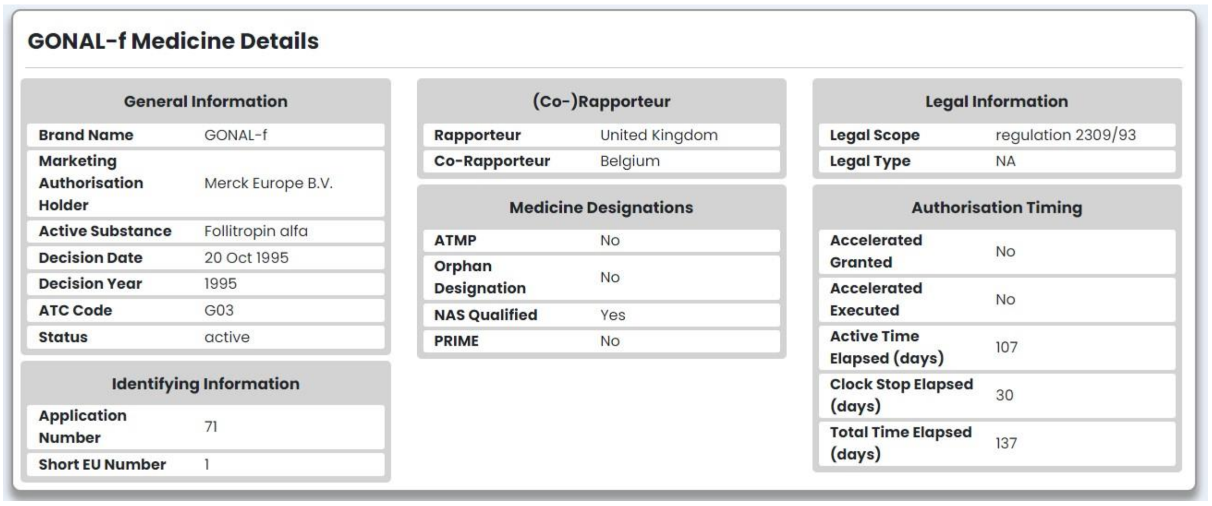
Short EU Number	Brand Name	Marketing Authorisation H	Decision Date	ATC Code
1	GONAL-f	Merck Europe B.V.	20 Oct 1995	G03
2	Taxotere	Sanofi Mature IP	27 Nov 1995	L01
3	Betaferon	Bayer AG	30 Nov 1995	L03
4	Fareston	Orion Corporation	14 Feb 1996	L02
5	CellCept	Roche Registration GmbH	14 Feb 1996	L04
6	NovoSeven	Novo Nordisk A/S	23 Feb 1996	B02
7	Humalog	Eli Lilly Nederland B.V.	30 Apr 1996	A10
8	Puregon	N.V. Organon	3 May 1996	G03
9	Zerit	Bristol-Myers Squibb Pharma EEIG	8 May 1996	J05
10	Rilutek	Sanofi Mature IP	10 Jun 1996	N07
11	Caelyx pegylated liposomal	Janssen-Cilag International NV	21 Jun 1996	L01
12	Bondronat	Atnahs Pharma Netherlands B.V.	25 Jun 1996	M05
13	Destara	Roche Registration Limited	25 Jun 1996	M05

Medicine specific data

Each data entry has an information sign, this links to a page containing information about that specific medicine. This data includes information about the various procedures of the medicine. These can be visualized in a timeline.



Decision Date	ATC Code
20 Oct 1995	G03
27 Nov 1995	L01
30 Nov 1995	L03
14 Feb 1996	L02
14 Feb 1996	L04
23 Feb 1996	B02
30 Apr 1996	A10
3 May 1996	G03
8 May 1996	J05
10 Jun 1996	N07
21 Jun 1996	L01
25 Jun 1996	M05
25 Jun 1996	M05



GONAL-f Medicine Details

General Information

Brand Name	GONAL-f
Marketing Authorisation Holder	Merck Europe B.V.
Active Substance	Follitropin alfa
Decision Date	20 Oct 1995
Decision Year	1995
ATC Code	G03
Status	active

Identifying Information

Application Number	71
Short EU Number	1

(Co-)Rapporteur

Rapporteur	United Kingdom
Co-Rapporteur	Belgium

Medicine Designations

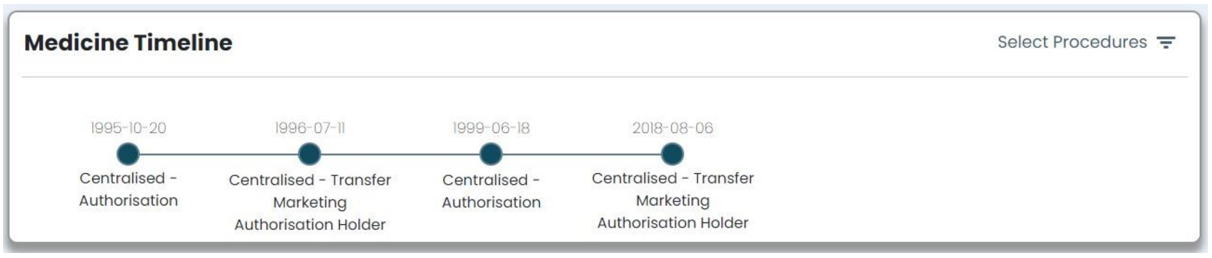
ATMP	No
Orphan Designation	No
NAS Qualified	Yes
PRIME	No

Legal Information

Legal Scope	regulation 2309/93
Legal Type	NA

Authorisation Timing

Accelerated Granted	No
Accelerated Executed	No
Active Time Elapsed (days)	107
Clock Stop Elapsed (days)	30
Total Time Elapsed (days)	137



Medicine Timeline

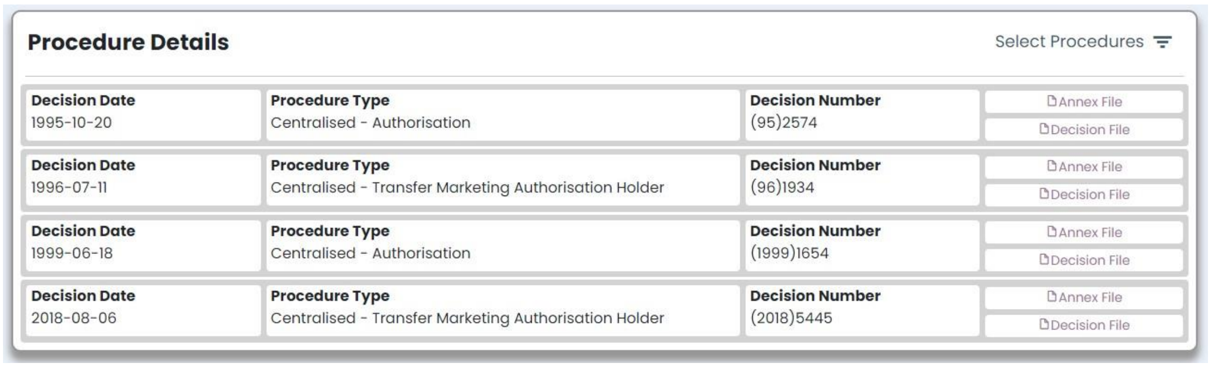
Select Procedures

1995-10-20 Centralised - Authorisation

1996-07-11 Centralised - Transfer Marketing Authorisation Holder

1999-06-18 Centralised - Authorisation

2018-08-06 Centralised - Transfer Marketing Authorisation Holder



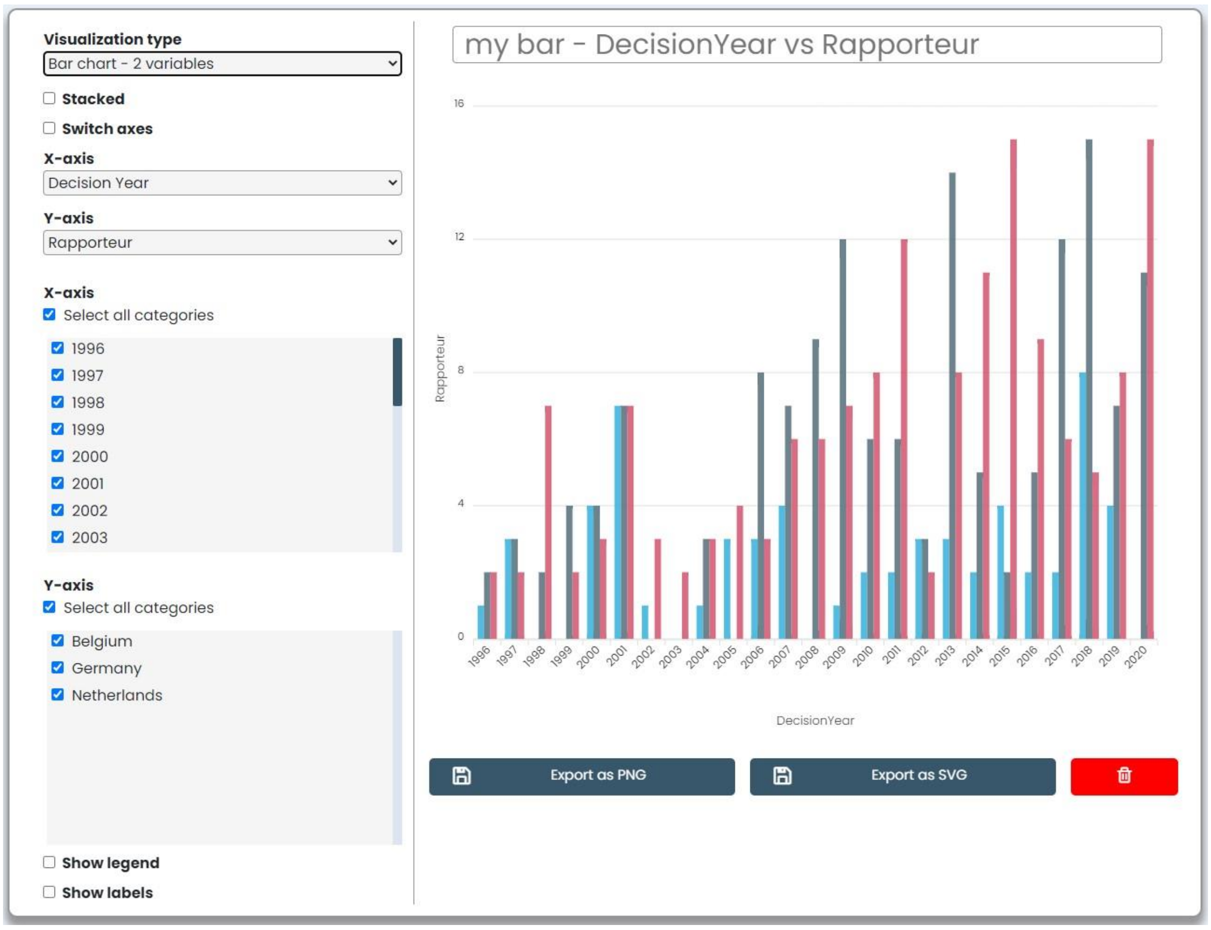
Procedure Details

Select Procedures

Decision Date	Procedure Type	Decision Number	
1995-10-20	Centralised - Authorisation	(95)2574	Annex File / Decision File
1996-07-11	Centralised - Transfer Marketing Authorisation Holder	(96)1934	Annex File / Decision File
1999-06-18	Centralised - Authorisation	(1999)1654	Annex File / Decision File
2018-08-06	Centralised - Transfer Marketing Authorisation Holder	(2018)5445	Annex File / Decision File

Visualizations page

On the visualizations page visualizations can be made based on the selected data.



Visualization type
Bar chart - 2 variables

☐ **Stacked**

☐ **Switch axes**

X-axis
Decision Year

Y-axis
Rapporteur

X-axis
☑ Select all categories
☑ 1996
☑ 1997
☑ 1998
☑ 1999
☑ 2000
☑ 2001
☑ 2002
☑ 2003

Y-axis
☑ Select all categories
☑ Belgium
☑ Germany
☑ Netherlands

☐ **Show legend**

☐ **Show labels**

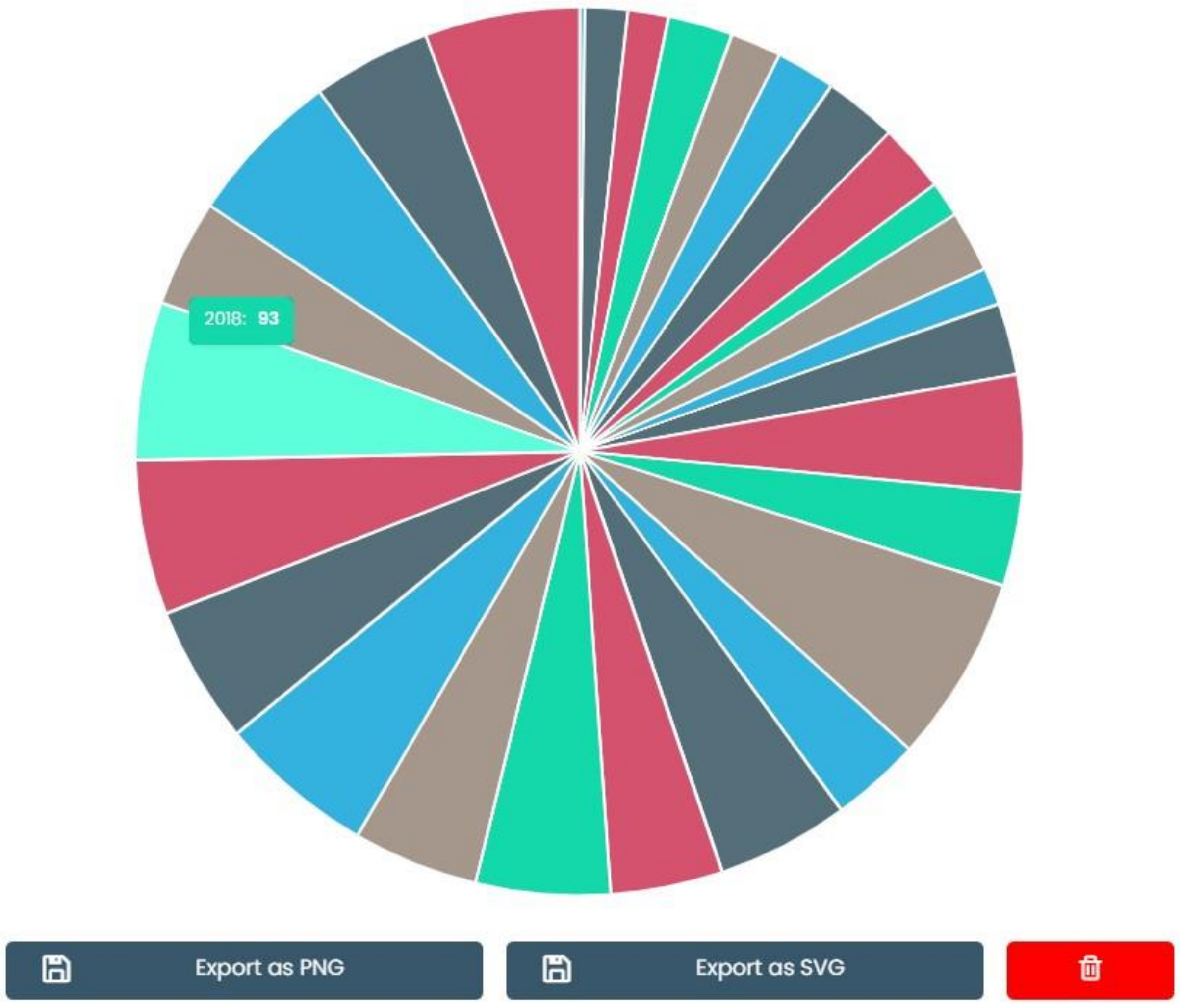
my bar - DecisionYear vs Rapporteur

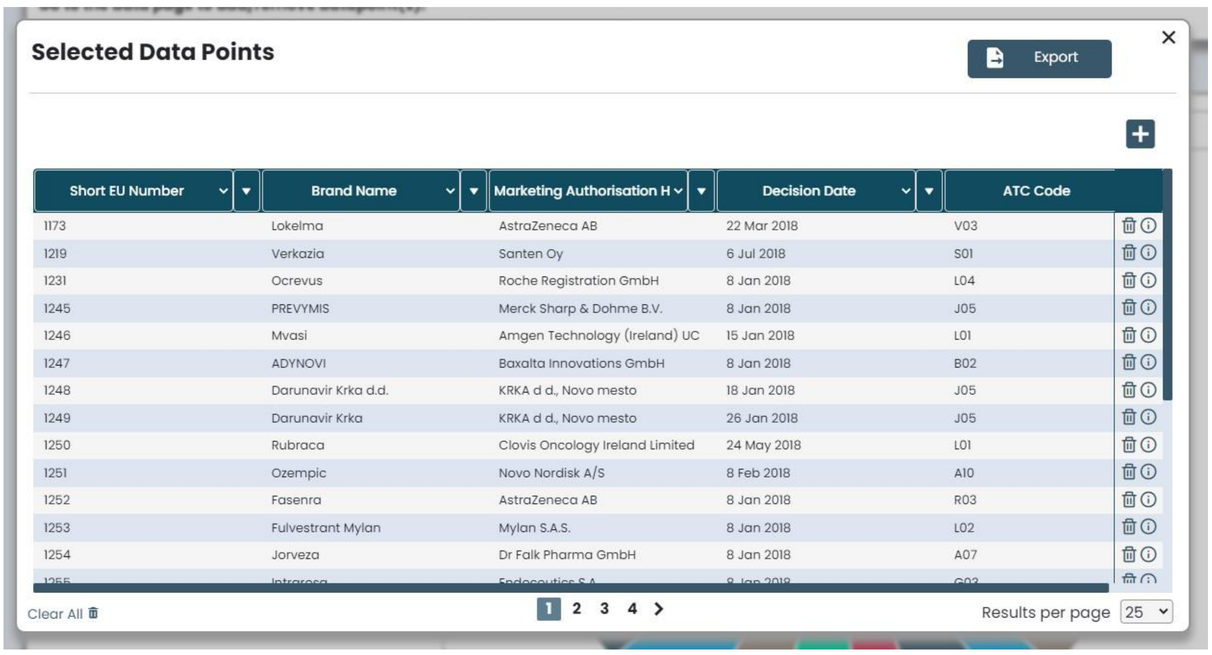
Visualizations

Multiple visualizations can be made (and subsequently removed). All visualizations made will use the same selected data. The user can choose between bar charts (with either 1 or 2 variables), line charts and pie charts. Each chart type can have their own options. All chart types have the ‘show labels’ and ‘show legend’ option. For each chart type you can choose any combination between variables. You can choose which categories of the chosen variables you wish to include. A bar chart with 2 variables has 3 extra options. It can switch its axes, make it stack and, if stacked, choose whether the stacking is relative. A title can be added, but currently it is not included when exported.

Interactivity

A visualization can be hovered over, showing information about the categories. In the line charts and single variable bar charts you can zoom in on the chart. When clicking on a category in a chart a popup is shown. This popup is essentially a shortcut to the lower table of the data page, containing the selected data, but only showing the data points of the selected category. In this popup, data entries can be exported or removed from the selection. When data entries are removed, all visualizations will be rerendered, because the selected data has been changed.





Selected Data Points

Export

Short EU Number	Brand Name	Marketing Authorisation H	Decision Date	ATC Code
1173	Lokelma	AstraZeneca AB	22 Mar 2018	V03
1219	Verkazia	Santen Oy	6 Jul 2018	S01
1231	Ocrevus	Roche Registration GmbH	8 Jan 2018	L04
1245	PREVYMIS	Merck Sharp & Dohme B.V.	8 Jan 2018	J05
1246	Mvasi	Amgen Technology (Ireland) UC	15 Jan 2018	L01
1247	ADYNOVI	Baxalta Innovations GmbH	8 Jan 2018	B02
1248	Darunavir Krka d.d.	KRKA d d., Novo mesto	18 Jan 2018	J05
1249	Darunavir Krka	KRKA d d., Novo mesto	26 Jan 2018	J05
1250	Rubraca	Clovis Oncology Ireland Limited	24 May 2018	L01
1251	Ozempic	Novo Nordisk A/S	8 Feb 2018	A10
1252	Fasenra	AstraZeneca AB	8 Jan 2018	R03
1253	Fulvestrant Mylan	Mylan S.A.S.	8 Jan 2018	L02
1254	Jorveza	Dr Falk Pharma GmbH	8 Jan 2018	A07
1255	Intrarosa	Endoceutics S.A.	8 Jan 2018	G03

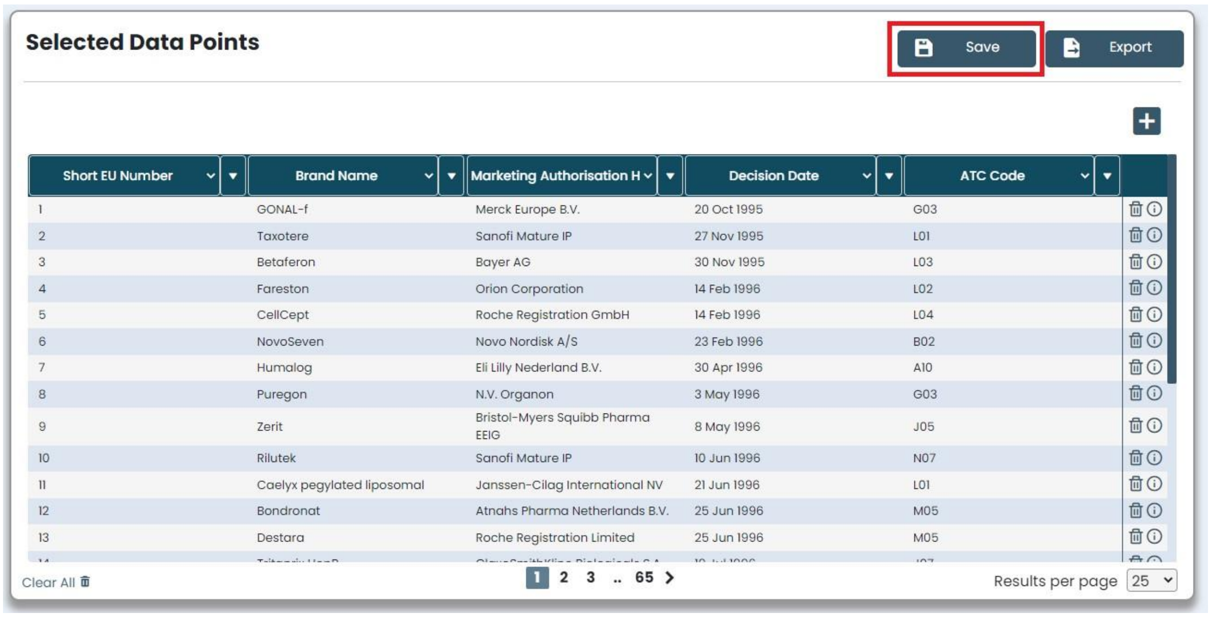
Clear All 1 2 3 4 Results per page 25

Exports

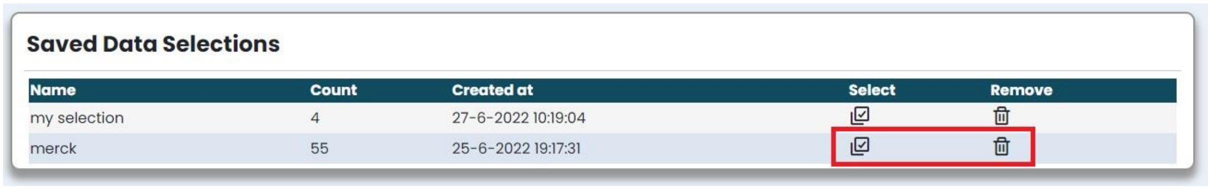
A visualization can be exported to either .svg or .png format.

Account

Users can login using the login button at the bottom of the navigation bar on the left side. A user can save data sets to their account, so they can easily access them later on. Saving a selection of the data is done on the data page and the button is placed next to the export button. Accessing a selection is done on the account page where saved selections can also be removed.



Short EU Number	Brand Name	Marketing Authorisation H	Decision Date	ATC Code
1	GONAL-f	Merck Europe B.V.	20 Oct 1995	G03
2	Taxotere	Sanofi Mature IP	27 Nov 1995	L01
3	Betaferon	Bayer AG	30 Nov 1995	L03
4	Fareston	Orion Corporation	14 Feb 1996	L02
5	CellCept	Roche Registration GmbH	14 Feb 1996	L04
6	NovoSeven	Novo Nordisk A/S	23 Feb 1996	B02
7	Humalog	Eli Lilly Nederland B.V.	30 Apr 1996	A10
8	Puregon	N.V. Organon	3 May 1996	G03
9	Zerit	Bristol-Myers Squibb Pharma EEIG	8 May 1996	J05
10	Rilutek	Sanofi Mature IP	10 Jun 1996	N07
11	Caelyx pegylated liposomal	Janssen-Cilag International NV	21 Jun 1996	L01
12	Bondronat	Atnahs Pharma Netherlands B.V.	25 Jun 1996	M05
13	Destara	Roche Registration Limited	25 Jun 1996	M05



Name	Count	Created at	Select	Remove
my selection	4	27-6-2022 10:19:04		
merck	55	25-6-2022 19:17:31		

Code expansion guide

General

We have tried to make the dashboard as dynamic as possible by making it depend on the backend, meaning we do not have many hardcoded components. Therefore, for tasks like adding a new variable, we refer to the backend guide. If a new variable is added on the backend, then the frontend instantly supports this new variable.

If you want to write unit tests that use new variables, the only things you need to do is replace the `structServer` json file, which is located in the `src\json` folder, with the new structure data retrieved from the respective endpoint of the backend. This file essentially translates the backend key to a frontend version and includes some information like the data type. The same must be done for the `allServerData` json file, which is located in the same directory, but then should be replaced with the updated medicines data retrieved from the respective endpoint again of the backend. All files in this directory are used to mock the backend server which allows to unit test certain components that normally need these data from the server.

Text

Text on e.g. the information and home page can easily be changed as well. They are all located in their respective page files, e.g. `HomePage` in `src\pages\home`. The text in the header is located in the `src\core\header` directory in the `Header` file. Analogously, the text in the footer is located in the `src\core\footer` directory in the `Footer` file.

Visualizations

There are a variety of visualization types, each having its own options or properties in their display. To keep the visualizations modular and therefore extensible, a visualization is built of 3 individual components: visualization_type, data_interface and form. These are all located in the `src\pages\visualizations\single_visualization` directory. We will discuss each of them more indepth.

visualization_types

The `visualization_types` directory contains a separate file for each chart. This file holds a function-based component which initializes the settings of the chart and renders the chart. To add a new chart, a new file must be created in this directory. The structure of such a file is as follows:

```
import React from 'react'
import Chart from 'react-apexcharts'

// Replace "SomeChart" with the name of your chart component, e.g. "BarChart"
function SomeChart(props) {
  let settings = {
    options: {
      // Some chart specific options here
    }
```

```
      // Add the series to the settings
      // The series is passed as a property to the component
      series: props.series,
    }

    return (
      // Replace "charttype" with the type of the chart, e.g. "bar"
      <div className="med-vis-chart">
        <Chart
          options={settings.options}
          series={settings.series}
          type="charttype"
          height={700}
        />
      </div>
    )
  }
```

The series, stored in the settings variable, is the data that will be displayed in the chart. This is constructed with the data_interface corresponding to the specific chart. More on that later. For information on which specific options are available for each chart type, visit the documentation of ApexCharts.

data_interfaces

We can not blindly pass the medicines data retrieved from the API to display the data in the chart. Some sort of processing of this data must be done beforehand. This is where the data_interface comes into play.

The `data_interfaces` directory contains a separate file for each chart type. This file holds a JavaScript function that calculates the series for the specific chart. The parameter `settings` passed to this function must contain the settings specified by the user in the form and the complete medicines data.

The directory furthermore contains the directories `shared_one_dimension` and `shared_two_dimensions`. These directories contain files again with functions that help calculating the series. The functions of `shared_one_dimension` are used by any chart with only one dimension (e.g. pie and histogram), while the functions of `shared_two_dimensions` are used by any chart with two dimensions (e.g. bar and line).

The `shared_one_dimension` directory only contains the file `pollChosenVariable`. This file holds a function with the same name. Based on the passed data, the chosen x-axis (the variable chosen to display) and the chosen categories (values of the variable), it calculates the frequency for each chosen category in the data (e.g. Decision Year 1995 has a frequency of 3). The function returns an array with a dictionary `dict` that maps categories to frequencies and another dictionary `euNumbers` that maps categories to the eu numbers that contributed to the frequency. The `dict` dictionary is essentially the calculated series. The `euNumbers` dictionary is additionally returned to identify the data points again. If a user clicks on a specific area of the chart, we can filter the data on these numbers and display this in a separate table. An example of the content those dictionaries:

```
// Variable: Decision Year
// Chosen categories: 1995
// Maps categories to frequencies
dict = {
  1995: 3
}

// Maps categories to their eu numbers
// So the data with eu numbers 1, 2 and 3 have Decision Year 1995
euNumbers = {
  1995: [1, 2, 3]
}
```

The `shared_two_dimensions` directory contains multiple files. To calculate the series, first the `pollChosenVariable` function is used. It is very similar to the one in the `shared_one_dimension`, however, a chosen y-axis and the chosen categories on the y-axis must be passed this time as well. It calculates the frequency again of the chosen categories on the x-axis, but it also makes a distinction between the chosen categories on the y-axis now. An example of those dictionaries:

```
// X-axis: Decision Year
// Y-axis: Rapporteur
// Chosen categories x-axis: 1995
// Chosen categories y-axis: United Kingdom and France

dict = {
  1995: {
    "United Kingdom": 2,
    "France": 1
  }
}

euNumbers = {
  1995: {
    "United Kingdom": [1, 3],
    "France": [2]
  }
}
```

However, this format is not the expected format of the library yet. It expects a dictionary that maps each category chosen on the y-axis to an array of frequencies, with each frequency element corresponding to a chosen category on the x-axis. This will be translated by the `createSelectedSeries` function into a format like this:

```
// X-axis: Decision Year
// Y-axis: Rapporteur
// Chosen categories x-axis: 2004, 2005 and 2006
// Chosen categories y-axis: Belgium and Ireland

series = {
  "Belgium": [1, 3, 3],
  "Ireland": [2, 2, 2]
}

euSeries = {
  "Belgium": [[270], [299, 301, 302], [325, 330, 333]],
  "Denmark": [[284, 286], [306, 307], [354, 361]]
}
```

The `toSeriesFormat` function will then finalize the format by putting the `series` and `euSeries` together in one output value, which can then be used to display the data in the chart.

To add support for a new chart, a new file must be created in the `data_interfaces` directory. Depending on the dimension of the chart, you should use the corresponding helper functions described above. The expected format of the series for a chart can be found again at the documentation of ApexCharts.

On top of that, a case to the switch statement in the `generateSeries` function must be added. The file is located in the `single_visualization\utils` directory. This function calls the series function which corresponds with the chart type. Add the following code to the switch:

```
// Replace "charttype" with the type of the chart, e.g. "bar"
case 'charttype':
  // Replace "generateChartSeries" with your new function that calculates the
series, e.g. "generateBarSeries"
  return generateChartSeries(settings)
```

forms

The `forms` directory contains the file `VisualizationForm` which is a function-based component that renders the main form of the visualizations page. The directory also contains the directories `types` and `shared`. The `shared` directory contains the `CategoryOptions` file. This file holds a function-based component rendering the category list, where categories can be selected based on the selected variable. This component is currently used in the forms of all the chart types. The `types` directory contains a separate file for each chart type. This file contains a function-based component that renders all the options on the form, e.g. "stacked" and "switch axes" for a bar chart, the dropdowns for the variables and the category options.

To add support for a new chart, a new file must be created in the `types` directory with similar content to the content of the other files. Based on what options this new chart has, you can build the modifiers in this new file. The options of a chart can be found again at the documentation of ApexCharts. When you have created such a new file, you should also add a new case to the switch statement in the `renderChartOptions` function in the `VisualizationForm` file:

```
// Replace "charttype" with the name of the new chart type, e.g. "bar"
case 'charttype':
  return (
    // Replace "ChartForm" with the name of your form component, e.g. "BarForm"
    <ChartForm
      uniqueCategories={props.uniqueCategories}
```

```
        onChange={handleChange}
        chartSpecificOptions={settings.chartSpecificOptions}
      />
    )
```

To make the new chart visible to the "Visualization type" dropdown, you should add an option to the select element in the return statement of the same file:

```
<!-- Replace "charttype" with the same name you just provided to the case in the
switch statement, e.g. "bar" -->
<option value="charttype">Your new chart type</option>
```

Add render

If you performed all the previous steps correctly, you should now be able to add the chart to the renderer. To do this, go back to the `single_visualization` directory. This contains the `SingleVisualization` file which holds a function-based component that renders the chosen visualization type and the export and remove buttons. Import the chart file you added in the `visualization_types` folder before and add a new case to the switch statement in the `renderChart` function:

```
// Replace "charttype" with the name of the new chart type, e.g. "bar"
case 'charttype':
  return (
    // Replace "SomeChart" with the name of your chart component, e.g. "BarChart"
    <SomeChart
      legend={legendOn}
      labels={labelsOn}
      id={id}
      series={series}
      categories={categories}
      options={options}
      onDataClick={onDataClick}
    />
  )
```

You may also want to add an extra case to the `renderTitlePlaceHolder` function in the `SingleVisualization` file to customize the title placeholder. The dashboard should now have support for your new visualization type.

Pages

Adding more pages to the dashboard is also possible. First, you should create a new directory in the `src\pages` directory with the name of your new page, e.g. `data`. In your new directory, a JavaScript file should be created with the appropriate name, e.g. `DataPage`. You can also create CSS file with the same name if you want to apply styling. The JavaScript file will hold a function-based component that renders your new page. It has the following structure:

```
import React from 'react'

// Import CSS for styling
import './YourPage.css'

function YourPage() {
  // Some variables and/or functions here
  return (
    // The HTML code of your new page (with CSS references)
  )
}

export default YourPage
```

Note that all pages in the dashboard are constructed this way. So making changes to the content of existing pages should be straightforward.

Then navigate to the `src\core` directory. This directory contains the `Routing` file holding a function-based component that provides the dashboard with routing information. To add your new page to the routing, add the following line between the `Routes` element in the return statement of the function:

```
// Replace "/PATH" with your desired path
// Replace "YourPage" with the name of the page component
<Route path="/PATH" element={<YourPage />} />
```

If you now enter your path in the url of the application, your new page should appear. However, the navigation bar does not show this new page yet. Navigate to the `src\core\navigation` directory. This directory contains the file `Navigation` which renders the navigation bar and its components. To add your new page to the navigation bar, add the following component between the other similar components in the return statement:

```
// Replace "NAME OF YOUR PAGE" with the name of your new page, e.g. "Data"
// Replace "CLASSNAME OF ICON" with the classname of a icon that suits your page;
see boxicons.com
// Replace "/PATH" with your earlier specified path

<NavLink
  name="NAME OF YOUR PAGE"
  image="CLASSNAME OF ICON"
  dest="/PATH"
  parent={this}
/>
```

Now the dashboard fully supports this new page.

Contexts

The application uses contexts to provide components with data. Data is regularly passed as properties in a component, but in some cases, the same data is used in multiple (non-related) components which makes it inconvenient to pass them constantly as properties to each other. It also allows to keep the state consistent while navigating to different pages.

To add a context, first navigate to the `src\shared\contexts` directory. This directory contains all the separate contexts of the application. Create a new file in this directory (which also ends with Context). The structure of this file is as follows:

```
import React from 'react'

// Replace "YourContext" with the name of your context, e.g. "DataContext"
const YourContext = React.createContext()

// Replace "useYour" with something that corresponds to the name, e.g. "useData"
export function useYour() {
  // Replace "YourContext" with the name of your context specified in the constant
  return useContext(YourContext)
}

// Replace "YourProvider" with the name of your context, e.g. "DataProvider"
export function YourProvider({ children }) {
  // Some variables and/or functions here
  return (
    // Replace "YourContext" with the name of your context specified in the
constant
    // Replace "InsertDataHere" with the variable that contains the data you want
to pass
    <YourContext.Provider value={InsertDataHere}>
      {children}
    </YourContext.Provider>
  )
}
```

What remains is adding this individual provider to the chain of providers. Navigate to the `src\shared` directory. This directory contains a `Provider` file which represents the chain of providers. You can add your provider somewhere between those other providers. It does not matter where you put it in general, but if you need data from one of other providers in your context, then you should put your provider as a child of this provider.

Now you can get this data anywhere in the application if you call the following statement:

```
// Replace "useYour" with the same name of the function of your context specified
earlier
const data = useYour()
```

Backend User Manual

Development set-up

This setup guide is written with Ubuntu 20.04 in mind. Most of the steps will be the same for other operating systems. There are a few prerequisites that we assume are already setup. These include:

- A Python3.10 install
- Access to a MySQL database

If the prerequisites are installed you can follow the guide below to get started.

In all the snippets below the current directory is medctrl-backend/ unless specified otherwise.

Install dependencies

0. Create and activate Python virtual environment.

```
python3 -m virtualenv venv
source venv/bin/activate
```

For windows use venv/bin/activate instead of source venv/bin/activate 1.

Install requirements.

```
pip install -r requirements.txt
```

During the installation of mysqlclient an error might occur because some libraries are not yet installed. These can be installed with the following command:

```
sudo apt install python3-dev default-libmysqlclient-dev build-
essential
```

Setup Configuration

The configuration files for the software should be placed in medctrl-backend/API/api_settings /settings/. There already is a settings file (common.py) with general settings that are the same for every deployment.
In addition to these settings you will need some extra configuration values, for example database login credentials. For development you should create a file named dev_settings.py in the settings/ directory. Below is an example configuration which you can use as a template.

```
# Import all common settings
from api_settings.settings.common import *

# Secret key that Django uses for generating hashes
SECRET_KEY = "random secret key"

# Whether to run Django in Debug mode
DEBUG = True

# Root path where the API will be listening on
BASE_URL = "api/"

# Database connection settings
# The DB credentials can be hardcoded or loaded in
# via environment variables
DATABASES = {
    "default": {
        "ENGINE"  "django.db.backends.mysql",
        "NAME"  DB_NAME,
        "USER"  DB_USERNAME,
        "PASSWORD"  DB_PASSWORD,
        "HOST"  DB_HOST,
        "PORT"  DB_PORT,
      }
}

# Where to put static files for django. Not needed for development
STATIC_ROOT = "django-static"
```

MedCtrl Setup

1. Migrate database and create Django permission levels

```
# in directory medctrl-backend/API
python manage.py migrate
python manage.py create_column_permissions
```

2. Create a Django admin user and anonymous group.
   To actually get data back, you need to assign permissions to this anonymous group. This can be done via the Django admin panel.
   More details about permissions can be found in the 'Managing Groups' section of this manual.

```
# in directory medctrl-backend/API
# this command will give some prompts for username and password
python manage.py createsuperuser
python manage.py init_setup
```

3. Run the backend

```
# in directory medctrl-backend/API
# Django will start on port 8000
python manage.py runserver 8000
```

Endpoints

- GET /medicine
  This endpoint is used to retrieve the medicine data. A GET request to this endpoint will return all the data that the (anonymous) user has access to.

- GET /procedure/{eunumber} This endpoint is used to retrieve all the procedures that are connected to the medicine with the given eunumber.

- /saveselection

  This endpoint is used to create, retrieve and delete saved selections.

  - Creating a saved selection
    POST /saveselection with following body:

~~~
```json
{
      "name": "Name of selection",
      "eunumbers": [1, 2, 3] // list of selected medicines
}
```

->

```json
{
      "id": "12fb0250-a725-462d-8b06-92762194a2af", // id of the
saved selection
      "name": "test",
      "created_at": "2022-06-15T11:11:59.466733Z",
      "created_by": "<username>",
      "eunumbers": [
            1,
            2,
          3
       ]
}
```
~~~

- Retrieving a single saved selection
  GET /saveselection/<selectionid>
  ->

```
{
    "id": "12fb0250-a725-462d-8b06-92762194a2af", // id of the saved
selection
    "name": "test",
    "created_at": "2022-06-15T11:11:59.466733Z",
    "created_by": "<username>",
    "eunumbers": [
        1,
        2,
        3
      ]
}
```

- Retrieving all saved selection for the current user
  GET /savedselection
  ->

```
[
      {
        "id": "12fb0250-a725-462d-8b06-92762194a2af", // id of the
saved selection
        "name": "test",
        "created_at": "2022-06-15T11:11:59.466733Z",
        "created_by": "<username>",
        "eunumbers": [
            1,
            2,
            3

      }
]
```

- Deleting a saved selection

-
  DELETE /saveselection/<selectionid>

  GET /detailedData

  This endpoint returns details about the types of variables that are in the database. This data is, among other things, used to determine how to filter/sort the medicine data.

- POST /account/login Used to login a user. POST with following body:

```
{
    "username":  username"
    "password":  password"
}
```

->

```
{
    "expiry": "2022-06-15T21:11:51.454234Z",
    "token":
"1c5cd1a9cacb2a883059d1b835378eee71adb3112f61cf38fb88252cbdcdd026"
    "user": {
        "id": 2,
        "username": "username",
        "groups": [

                "name": "group1",
                "id": 1

          /}, list of groups that the user is in
        "selections": [] // list of saved selection
      }
}
```

- POST /account/logout, /account/logoutAll

  These endpoints are used to logout the user. The logoutAll will end all active sessions of the given user. To logout, send a POST request with an empty body to one of these URLs.

  POST /scraper/medicine This endpoint can be used to programatically (for example via the scraper) update the medicine data in de database.

-

  Updates can be done by sending a POST request with the following body\

  ```
  {
      "override": true, // Whether to override manually updated values
      "data": [

              "eunumber": 1,

            //} A list of medicine objects that will be updated
            // For a complete list of variables that can be updated
            // See the domain model
         ]
  }
  ```

- POST /scraper/procedure This endpoint can be used to programatically (for example via the scraper) update the procedure data in de database.

  Updates can be done by sending a POST request with the following body

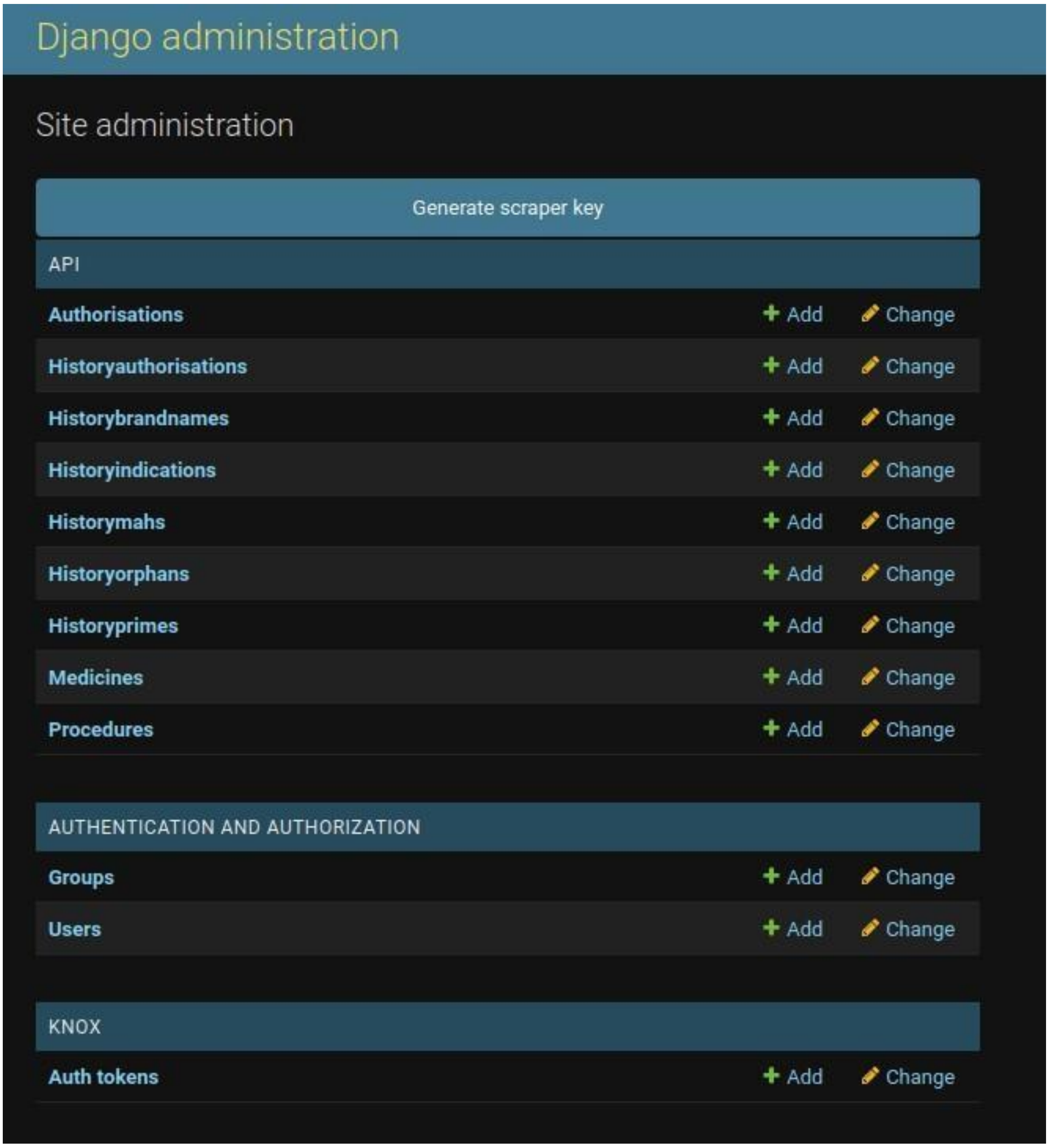
  ```
  {
      "override": true, // Whether to override manually updated values
      "data": [

              "eunumber": 1,
              "procedurecount": 1,

            //} A list of procedure objects that will be updated
            // For a complete list of variables that can be updated
            // See the domain model
         ]
  }
  ```

Admin panel

The Django admin panel is accessible via the <ROOT_URL>/admin/ endpoint.



Django administration

Site administration

Generate scraper key

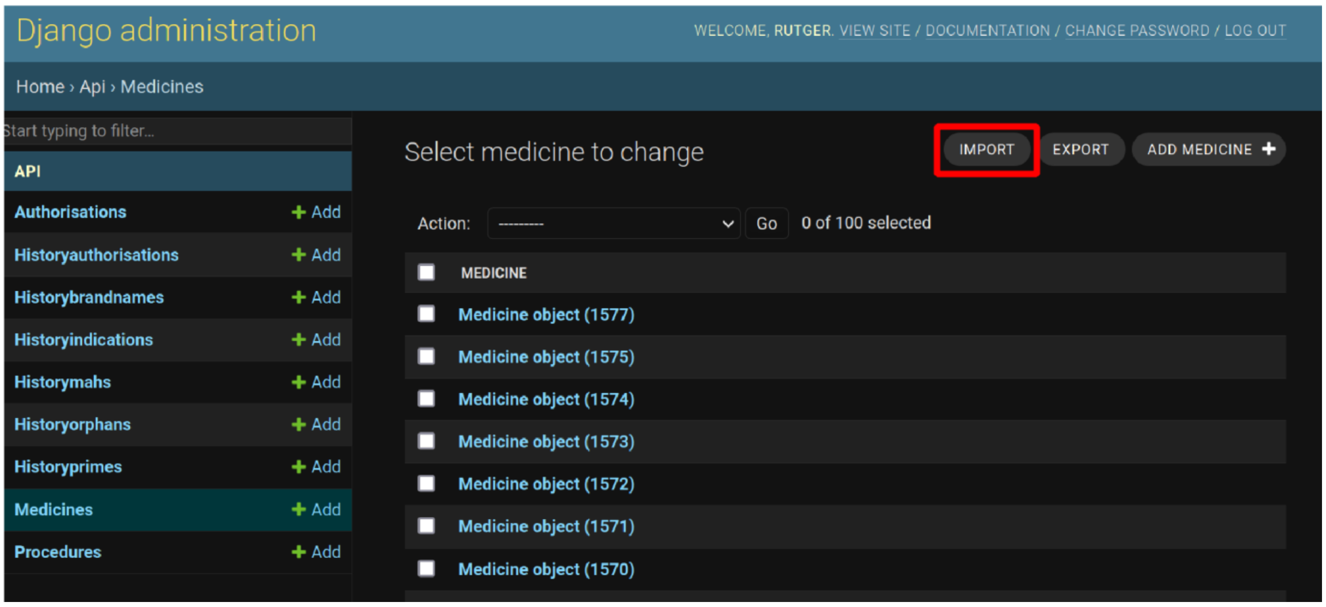
API		
Authorisations	Add	Change
Historyauthorisations	Add	Change
Historybrandnames	Add	Change
Historyindications	Add	Change
Historymahs	Add	Change
Historyorphans	Add	Change
Historyprimes	Add	Change
Medicines	Add	Change
Procedures	Add	Change

AUTHENTICATION AND AUTHORIZATION		
Groups	Add	Change
Users	Add	Change

KNOX		
Auth tokens	Add	Change

Importing data from existing Excel sheets

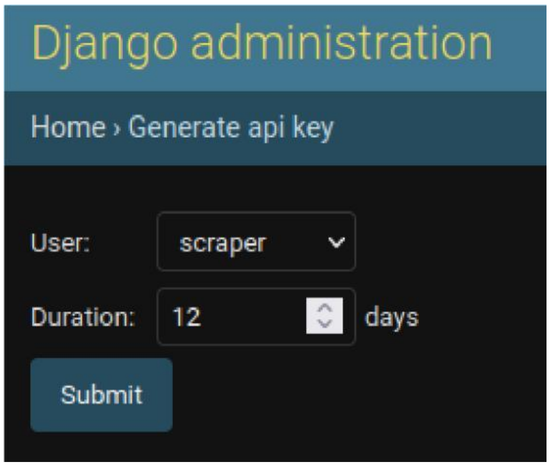
We have curated the initial dataset into Excel files that can be imported to the system.
The data files can be found at medctrl-backend/curated_data/. Clicking on a model in the Admin panel will show import and export options in the top-right:

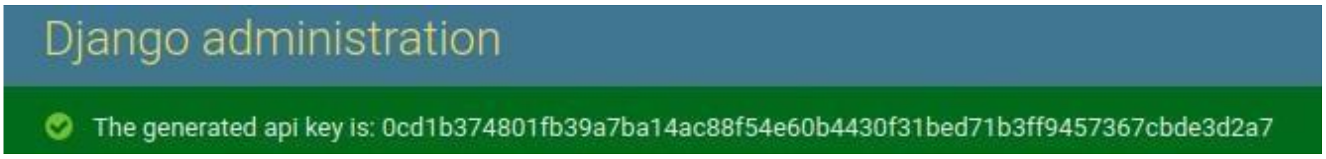


After submitting a file you will get an overview of the data that will be imported. You can then click confirm to actually import the data.
Please note that importing the Procedures can take quite some time because it is a lot of data. (the 41k procedures took around 2-3 minutes to imoprt on a basic VM)

Scraper API keys

In the admin panel you can generate an API key for the scraper. You can specify how many days the API key should be valid for and upon clicking generate you will be given the token which can be copied to the scraper.

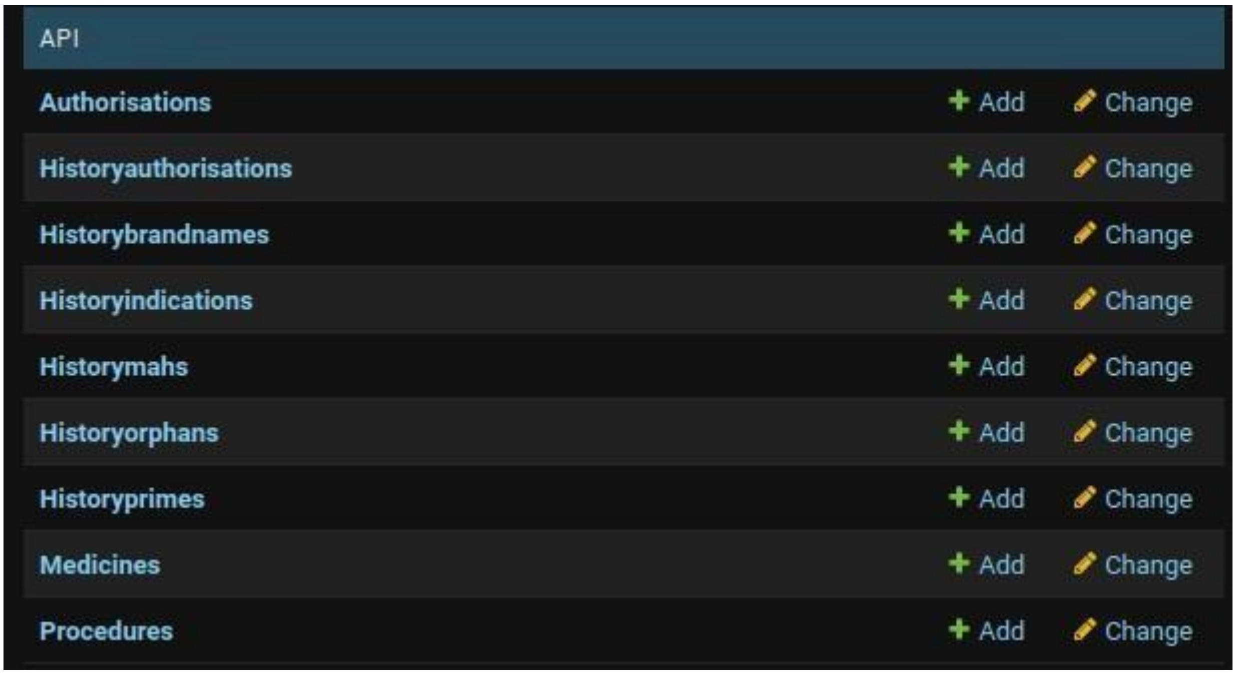




The generated api key is: 0cd1b374801fb39a7ba14ac88f54e60b4430f31bed71b3ff9457367cbde3d2a7

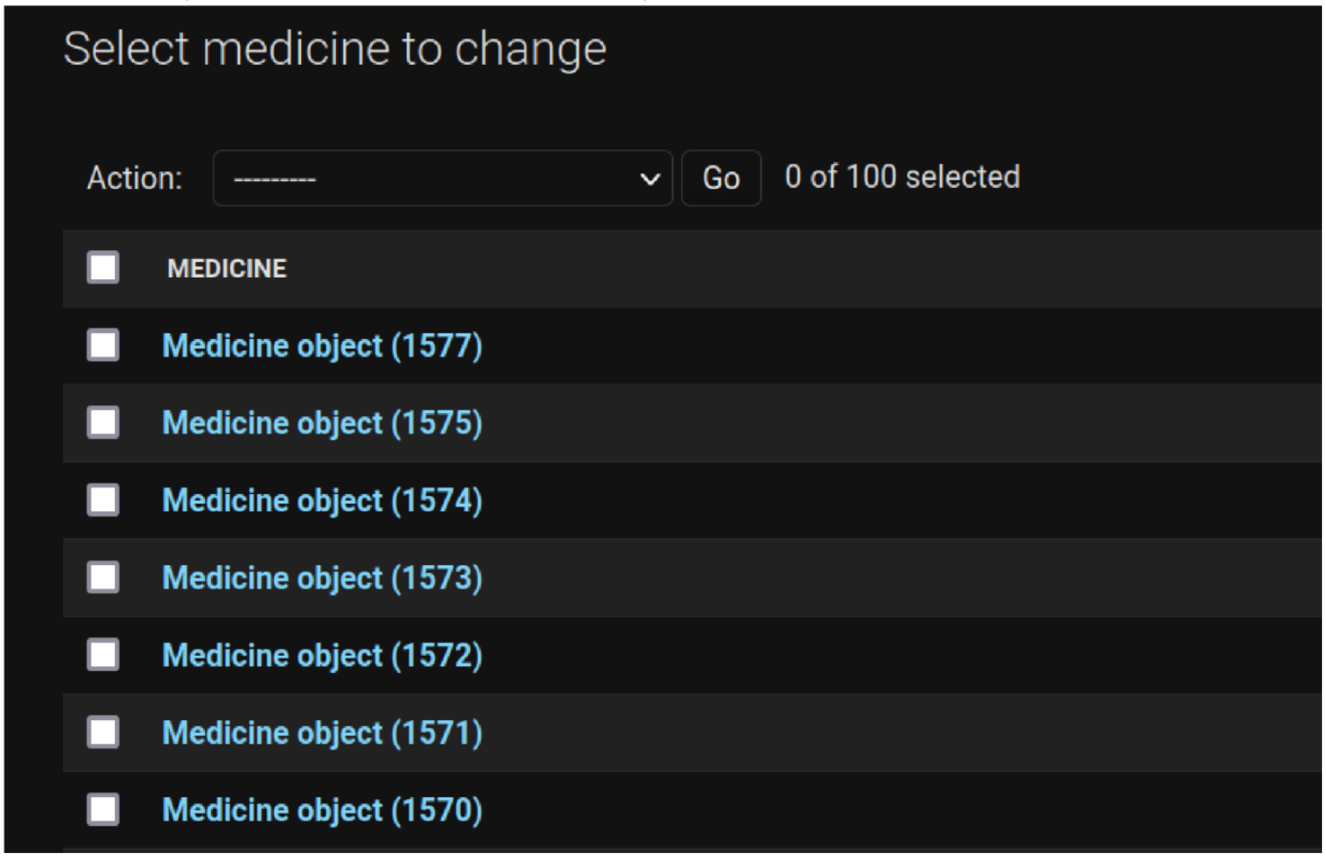
Manually update model values

It is possible to update individual objects values in the admin panel by following the links in the overview.



API		
Authorisations	+ Add	Change
Historyauthorisations	+ Add	Change
Historybrandnames	+ Add	Change
Historyindications	+ Add	Change
Historymahs	+ Add	Change
Historyorphans	+ Add	Change
Historyprimes	+ Add	Change
Medicines	+ Add	Change
Procedures	+ Add	Change

You will then be presented with an overview of all the objects that are in the database.

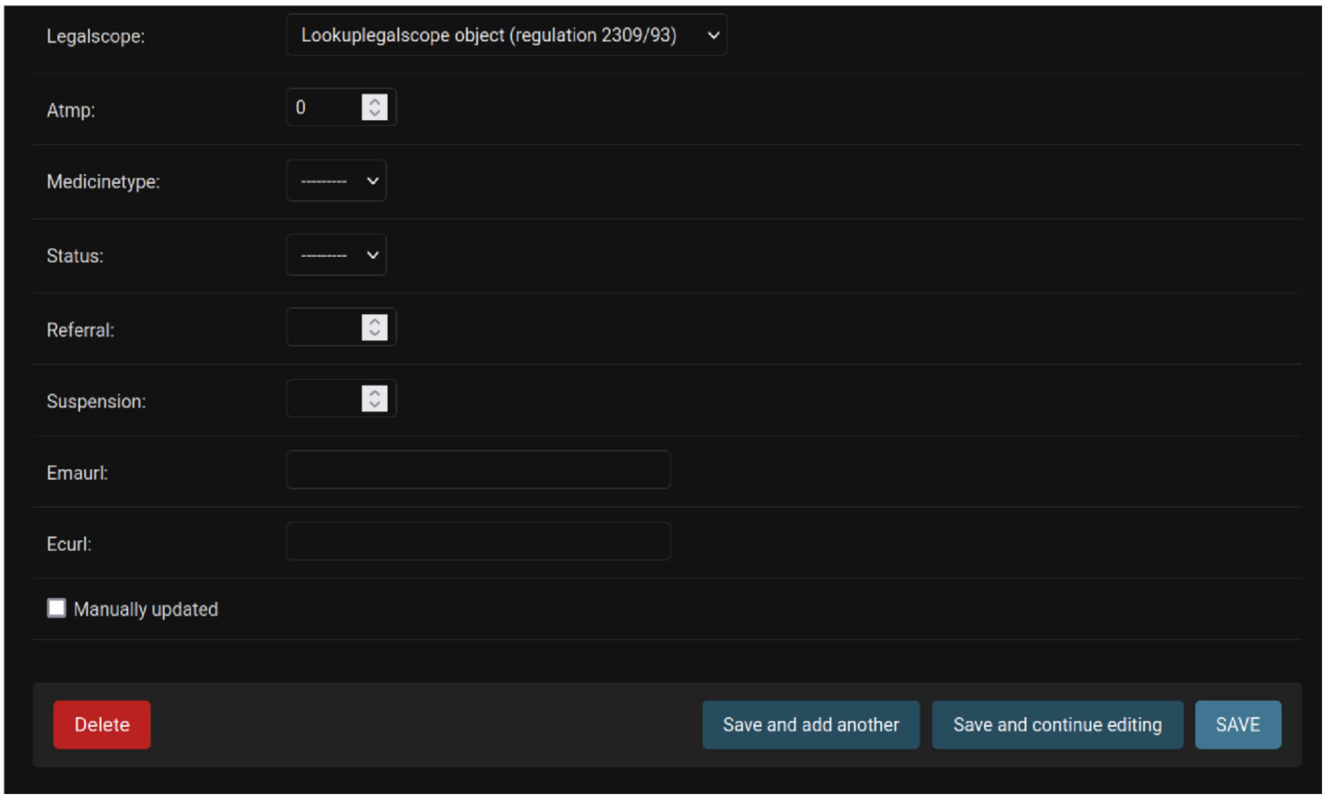


Select medicine to change

Action: --------- Go 0 of 100 selected

MEDICINE
Medicine object (1577)
Medicine object (1575)
Medicine object (1574)
Medicine object (1573)
Medicine object (1572)
Medicine object (1571)
Medicine object (1570)

Clicking on a medicine object will give options to edit or delete the object.

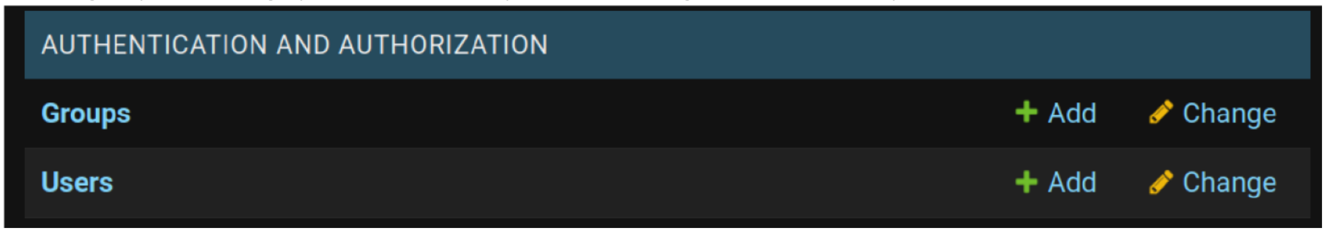


Legalscope:	Lookuplegalscope object (regulation 2309/93)
Atmp:	0
Medicinetype:	---------
Status:	---------
Referral:	
Suspension:	
Emaurl:	
Ecurl:	

☐ Manually updated

Group management

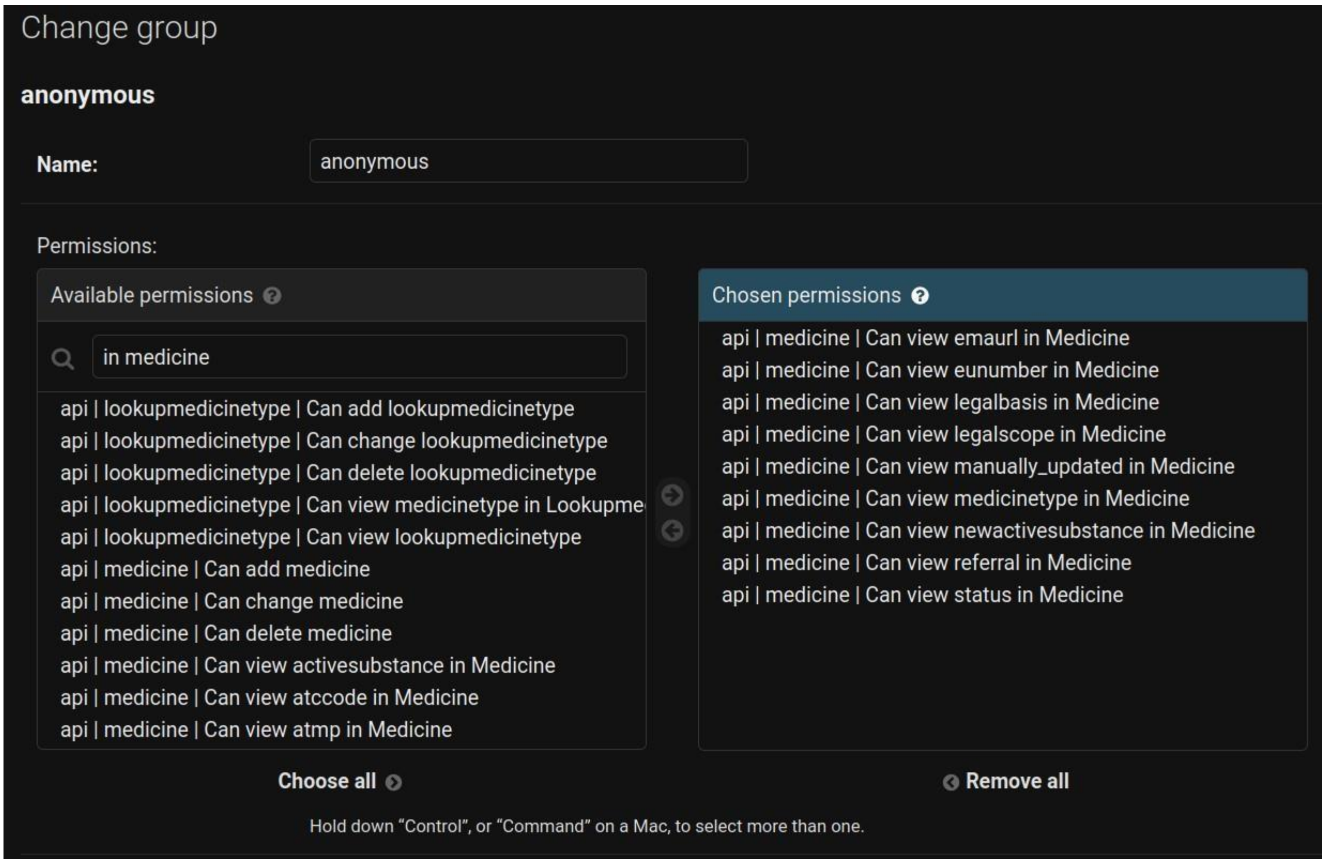
We use groups to manage permissions. Groups can be managed via the admin panel.



AUTHENTICATION AND AUTHORIZATION

Groups	+ Add	Change
Users	+ Add	Change

By default the initial setup script will create an anonymous group. If someone who is not logged in sends a request, the anonymous groups' permissions will be used to determine what data to send back. In the edit menu for a group you can specify which variables of a medicine a specific group has access to.


Change group

anonymous

Name: anonymous

Permissions:

Available permissions

in medicine

api | lookupmedicinetype | Can add lookupmedicinetype
api | lookupmedicinetype | Can change lookupmedicinetype
api | lookupmedicinetype | Can delete lookupmedicinetype
api | lookupmedicinetype | Can view medicinetype in Lookupme
api | lookupmedicinetype | Can view lookupmedicinetype
api | medicine | Can add medicine
api | medicine | Can change medicine
api | medicine | Can delete medicine
api | medicine | Can view activesubstance in Medicine
api | medicine | Can view atccode in Medicine
api | medicine | Can view atmp in Medicine

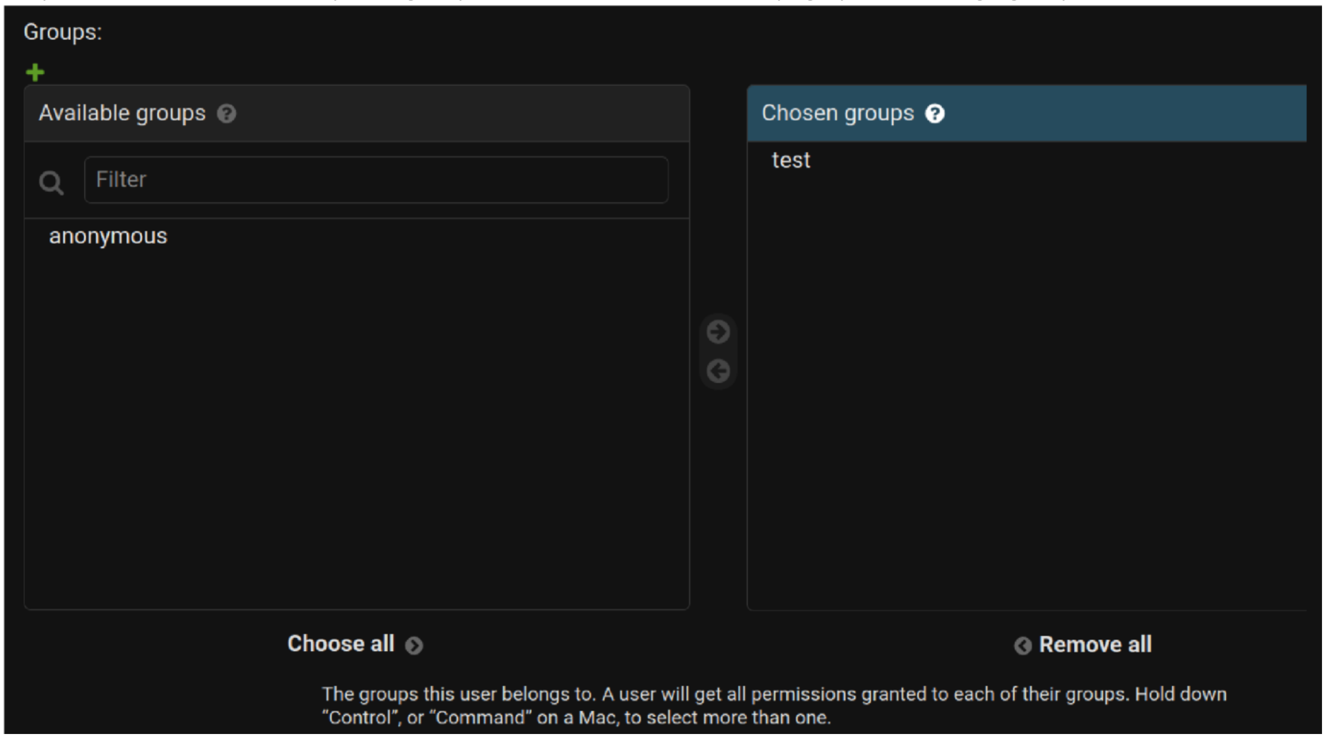
Chosen permissions

api | medicine | Can view emaurl in Medicine
api | medicine | Can view eunumber in Medicine
api | medicine | Can view legalbasis in Medicine
api | medicine | Can view legalscope in Medicine
api | medicine | Can view manually_updated in Medicine
api | medicine | Can view medicinetype in Medicine
api | medicine | Can view newactivesubstance in Medicine
api | medicine | Can view referral in Medicine
api | medicine | Can view status in Medicine

Choose all **Remove all**

Hold down “Control”, or “Command” on a Mac, to select more than one.

It is possible to create as many new groups as needed. On the users' page you can assign groups to users.


Groups:

Available groups

Filter

anonymous

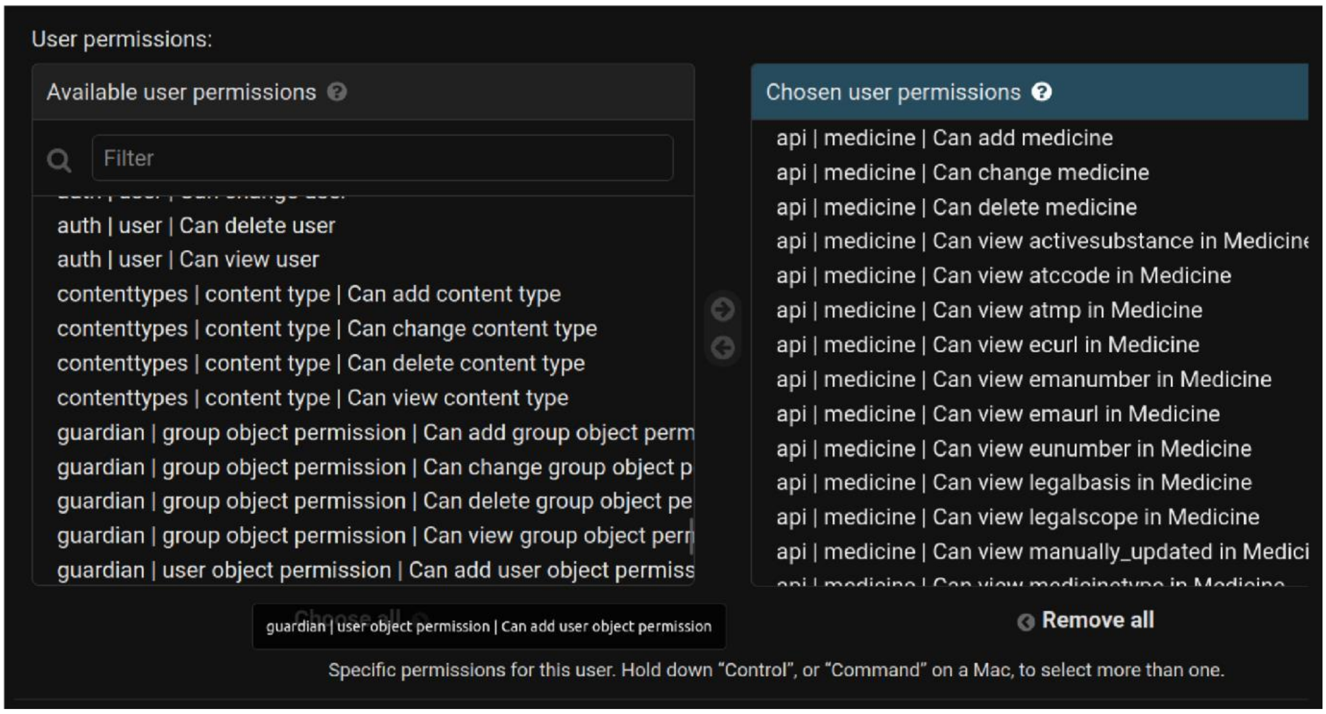
Chosen groups

test

Choose all **Remove all**

The groups this user belongs to. A user will get all permissions granted to each of their groups. Hold down “Control”, or “Command” on a Mac, to select more than one.

It is also possible to assign permissions to individual users.



User permissions:

Available user permissions

Filter

auth | user | Can delete user
auth | user | Can view user
contenttypes | content type | Can add content type
contenttypes | content type | Can change content type
contenttypes | content type | Can delete content type
contenttypes | content type | Can view content type
guardian | group object permission | Can add group object perm
guardian | group object permission | Can change group object p
guardian | group object permission | Can delete group object pe
guardian | group object permission | Can view group object per
guardian | user object permission | Can add user object permiss

guardian | user object permission | Can add user object permission

Chosen user permissions

api | medicine | Can add medicine
api | medicine | Can change medicine
api | medicine | Can delete medicine
api | medicine | Can view activesubstance in Medicine
api | medicine | Can view atccode in Medicine
api | medicine | Can view atmp in Medicine
api | medicine | Can view ecurl in Medicine
api | medicine | Can view emanumber in Medicine
api | medicine | Can view emaurl in Medicine
api | medicine | Can view eunumber in Medicine
api | medicine | Can view legalbasis in Medicine
api | medicine | Can view legalscope in Medicine
api | medicine | Can view manually_updated in Medici

Specific permissions for this user. Hold down “Control”, or “Command” on a Mac, to select more than one.

Adding variables

To add a new variable to the system there are a few things that need to be done:

- Add a new column to the appropriate model:
  models are located in the API/api/models/ folder.
  To add a variable with name X to medicine for example you need to add a field X to the the Medicine model class in medicine.py. For more details on what types or relations are available please see the Django documentation

- Add an entry to API/views/other/medicine_info_json.py

In here you should specify what type the variable has and what the displayname should be. This is, among other things, used by the frontend to determine how to sort on this variable. An example entry would be:

```
{
    "data-key": "X",
    "data-front-key": "X",
    "data-format": "string",
    "data-value": "Display name for frontend"
}
```

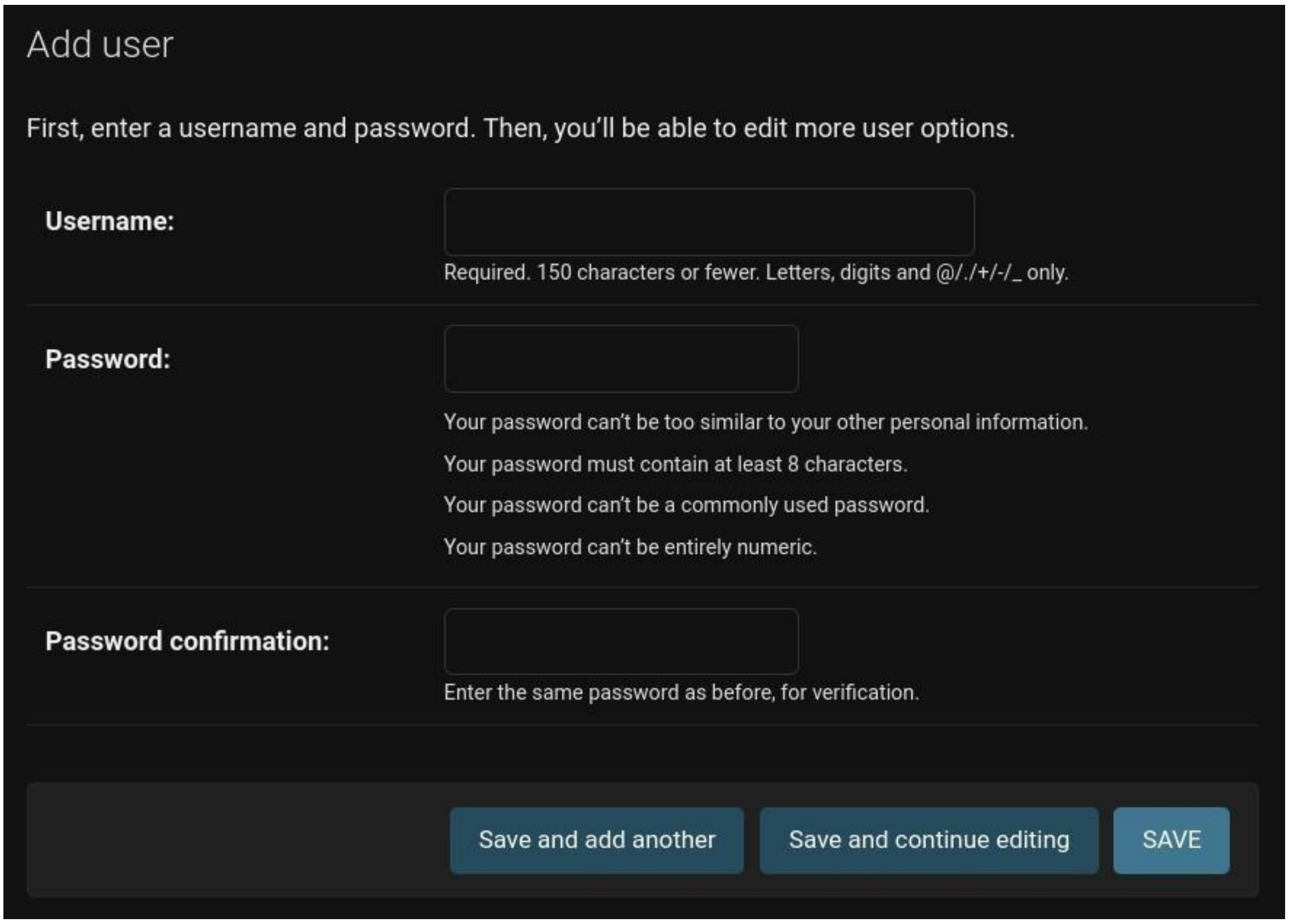
- Create a new migration and migrate the database

  Run the following commands to migrate the new changes:

```
# in directory medctrl-backend/API
python manage.py makemigrations
python manage.py migrate
python manage.py create_column_permissions
```

User management

Normal users can be created in the Django admin panel.


Add user

First, enter a username and password. Then, you'll be able to edit more user options.

Username:	Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.
Password:	Your password can't be too similar to your other personal information. Your password must contain at least 8 characters. Your password can't be a commonly used password. Your password can't be entirely numeric.
Password confirmation:	Enter the same password as before, for verification.

Once you have created a user you can assign groups and permissions to that user in the admin panel.

The response of sending a login request (more details can be found in the 'Endpoints' section of this manual) contains a token key. This token is used to authenticate follwing requests.

The token needs to be send in the Authorization header.

The value of the headers should be: Token <token>, where <token> is the value that was returned with the login response.

Scraper endpoint

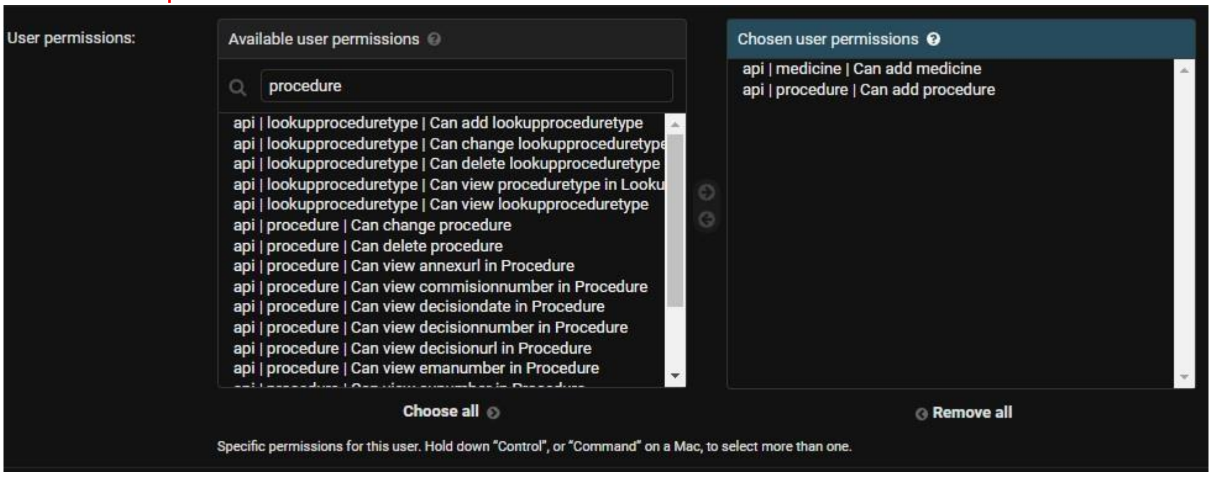
Scraper endpoints have been implemented in the backend of the medicine regulation database. These endpoints can be accessed by going to the baseurl of the backend followed by /scraper/medicine/ and /scraper/procedure/ so for example with a domain name of https://medctrl.nl/ and a base url of api/, the scraper endpoint for medicine will be https://medctrl.nl/api/scraper/medicine/. These endpoints can only be called by a post request with an authorized authentication key. The data contents of the post request should be a valid json format.

Authorization

The authorization of the scraper endpoints is handled by token authentication in the post request. This token authentication is coupled to an user which must have valid permissions for the scraper endpoints. To generate a token for the scraper user we first need to have a scraper user account. This account can not be a staff or admin user for security reasons.

Create scraper user

1. login to the admin panel ({baseurl}/admin/) using an admin (super) user (only a superuser can login to the admin panel)
2. go to Users
3. click on Add user
4. Add a username and password for the scraper user (Advised to use scraper in the username for clarity)
5. Click on Save
6. In the edit user screen (current screen, also accessible by clicking on the user), go to the user permissions table.
7. Add the permissions: “api | medicine | Can add medicine” and “api | procedure | Can add procedure” by pressing on the permission and pressing the arrow to the right. (You can also add only one of the permissions if you want the user to only have permission to one of the endpoints.)
   Warning: For security reasons it is advised to only add the permissions for the scraper and no other permissions



User permissions:

Available user permissions

procedure

api | lookupproceduretype | Can add lookupproceduretype
api | lookupproceduretype | Can change lookupproceduretype
api | lookupproceduretype | Can delete lookupproceduretype
api | lookupproceduretype | Can view proceduretype in Looku
api | lookupproceduretype | Can view lookupproceduretype
api | procedure | Can change procedure
api | procedure | Can delete procedure
api | procedure | Can view annexurl in Procedure
api | procedure | Can view commisionnumber in Procedure
api | procedure | Can view decisiondate in Procedure
api | procedure | Can view decisionnumber in Procedure
api | procedure | Can view decisionurl in Procedure
api | procedure | Can view emanumber in Procedure

Chosen user permissions

api | medicine | Can add medicine
api | procedure | Can add procedure

Specific permissions for this user. Hold down “Control”, or “Command” on a Mac, to select more than one.

8. The scraper user is now created with the right permissions

Generate authentication key for the scraper

Prerequisite: Have a scraper user with the right permissions. For the medicine endpoint use “api | medicine | Can add medicine”, for the procedure endpoint use “api | procedure | Can add procedure”

1. Go to the homepage of the admin panel
2. Press on the Generate scraper key button
3. Select the user with the scraper permissions (this dropdown will not display any staff and admin users)
4. Add a time for which the scraper key will be valid (182 days is advised, we think this is the right balance between security and the amount of replacing the key, minimum duration is 1 day, maximum is 365 days) after this day the key will expire and the scraper will not be authorized anymore.
5. Press Submit
6. The site will display a message with the generated key, this is the authentication token. Copy this key and use it as the authentication token of the request. (This key will only be generated once and can not be found later, if you lost the key you will need to generate a new key)
7. Add a reminder in your personal calendar to replace the scraper key a few days before the key expires, the scraper will not be able to access the endpoints when the key has expired.

Use authentication key in a request

Prerequisite: Have a key for the scraper user
Add the key to the html authorization header in the request using the following format:
'Authorization': 'Token {key}'
{key} is here the generated key for the scraper user. For another example see the python post request example paragraph.

Json format

All information is sent using the json data format (for more information see https://www.json.org/json-en.html). In this format we use 2 variables, override and data. The override variable is a setting, if this setting is true the endpoint will override all the information of the given medicines. If this setting is false, the scraper will only override null values and flexible variables (for an overview of the variables see the Data Information paragraph).
The second variable is the data variable. This is a list with all the medicines/procedures (depending on the endpoint). This is an example of a correct json format for the medicine endpoint:

```
{
 "override": false, "data":
 [
    {
   "eunumber": 1,
   "emanumber": "71",
   "atccode": "G03G",
   "activesubstance": "Follitropin alfa",
   "newactivesubstance": 0,
   "legalbasis": null,
   "legalscope": "Council Regulation (EEC) No 2309/93",
   "atmp": 0,
   "medicinetype": null,
   "status": "active",
   "referral": null,
   "suspension": null,
   "emaurl": "http: //www.ema.europa.eu/ema/index.jsp",
   "ecurl": "https: //ec.europa.eu/health/",
   "rapporteur": "United Kingdom",
   "corapporteur": "Belgium",
   "acceleratedgranted": 0,
   "acceleratedmaintained": 0,
   "authorisationtotaltime": 137,
   "authorisationactivetime": 107,
   "authorisationstoppedtime": 30,
   "decisionurl": null,
   "annexurl": null,
   "eparurl": null,
   "brandname": "Sutent",
   "mah": "Pfizer Europe MA EEIG",
   "orphan": 1,
   "prime": 0
    },
    {
   "eunumber": 1,
   "emanumber": "More examples like the one above can be added like this",
    }
 ]
}
```

And a correct example for the procedure endpoint:

```
{
  "override": false, "data":
  [
     {
    "eunumber": 1,
    "procedurecount": 1,
    "commisionnumber": 1,
    "proceduredate": "2012-04-23",
    "proceduretype": "sometype",
    "decisiondate": "2012-04-23",
    "decisionnumber": 1,
    "decisionurl": "http://dec.eu/url",
    "annexurl": "http://annex.eu/url"
     }
  ] }
```

Data information

Data sent to the scraper endpoints should have the following formats: All data which is not required can be null All boolean data values need to be sent as 0 or 1.

Medicine:

Name	Scraper name	Input type	Flexible	Required
eunumber	eunumber_abb	integer	No	Always
emanumber	emanumber	String	No	No
atccode	atccode	String	Yes	No
activesubstance	activesubstance	String	No	No
newactivesubstance	new_active_substance	{0, 1}	No	No
legalbasis	legal_basis	String	No	No
legalscope	legal_scope	String	No	No
atmp	ATMP	{0, 1}	No	No
medicinetype	-	String	No	No
status	group	String	Yes	No
referral	-	{0,1}	Yes	No
suspension	-	{0,1}	Yes	No
emaurl	emalink	String	Yes	No

ecurl	url	String	Yes	No
rapporteur	-	String	No	No
corapporteur	-	String	No	No
acceleratedgranted	-	{0,1}	No	No
acceleratedmaintained	-	{0,1}	No	No
authorisationtotaltime	-	Integer	No	No
authorisationactivetime	-	Integer	No	No
authorisationstoppedtime	-	Integer	No	No
decisionurl	-	String	Yes	No
annexurl	-	String	Yes	No
eparurl	-	String	Yes	No
brandname	brand_name	String	Yes (added to history)	No
mah	company	String	Yes (added to history)	No

orphan	orphan_drug	{0,1}	Yes (added to history)	No
prime	prime	{0,1}	Yes (added to history)	No

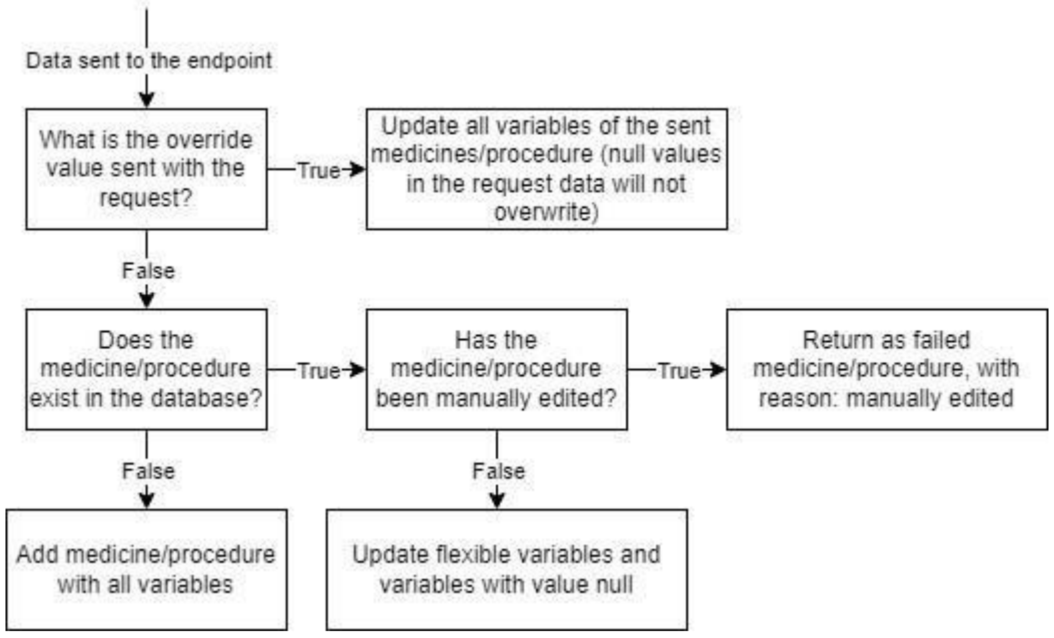
Procedure
For the procedure data we do not know the scraper names

Name	Scraper name	Input type	Flexible	Required
eunumber	-	Integer	No	Always
procedurecount	-	Integer	No	Always
commisionnumber	-	Integer	No	Yes (on addition)
emanumber	-	String	No	No
proceduredate	-	Date (YYYY-MM-DD)	No	No
proceduretype	-	String	No	No
decisiondate	-	Date (YYYY-MM-DD)	No	No
decisionnumber	-	Integer	No	No
decisionurl	-	String	Yes	No
annexurl	-	String	Yes	No

Scraper endpoint logic

Underneath you can find a diagram with the scraper endpoint logic. The failed updates/additions will be sent as a response with the reason why these actions failed.

The override value is a value sent in the request. See the Json format part of the manual.


Data sent to the endpoint
What is the override value sent with the request?
True
Update all variables of the sent medicines/procedure (null values in the request data will not overwrite)
False
Does the medicine/procedure exist in the database?
True
Has the medicine/procedure been manually edited?
True
Return as failed medicine/procedure, with reason: manually edited
False
Add medicine/procedure with all variables
False
Update flexible variables and variables with value null

Manually updated values cannot be updated through the API endpoints anymore. This is done so the information can be checked manually, to check if the information is correct. 2 things can be done to update these values:

1. Send a new request to the api with the override setting on true. This will bypass the restrictionof manually edited values and update all the variables regardless. This will also put the manually edited values back on false.
2. Update the values manually via the admin panel, if you want you can put the manually updatedvalue back on false in the admin panel.

Python post request example

This is a code example of how to implement the post request in python using the requests library. The variable scraper_data should be defined before this example and should have the format as described in the Json format paragraph.

```
#example of post request to scraper endpoints import
requests

API_ENDPOINT = "[input url to scraper endpoint here]"

API_KEY = "Token [input token key for api here]"

API_HEADERS={
   'Content-type':'application/json',
   'Accept':'application/json',
   'Authorization': API_KEY
}
```

```
# sending post request and saving response as response object
r = requests.post(url = API_ENDPOINT, headers= API_HEADERS, data = scraper_data)

#write failed rows to output file, (so this can be checked later) with
open('failedMedicines.txt', 'w') as f:
   f.write(r.text)
   f.close()
```

Permissions
Here is a list of all the possible permissions and where they grant access to. Not all permissions have a function in our application, these are provided by django rest framework. Superusers have all permissions by default.

**, xx and yy means any text standing there
Important permissions for the application

Name	**Function**	**Security Risk**
api | saved selection | **	Permissions for the saved selection, add, delete and view are used for saving data selection for a user	Low
api | medicine | can add medicine	used for adding medicines, also used for the medicine endpoint of the scraper	Medium
api | procedure | can add procedure	used for adding procedures, also used for the procedure endpoint of the scraper	Medium
api | yy | Can view xx in yy	Permission to view the variable xx from table yy	Low

All permissions available

Name	**Function**	**Security Risk**
admin | log entry | **	Admin log, not used in our application	Low

api | saved selection | **	Permissions for the saved selection, add, delete and view are used for saving data selection for a user	Low
auth | group | **	Change groups for authentication, not used in our application	Medium
auth | permission | **	Change permissions for user, not used in our application	High
contenttypes | **	Not used in our application	Low
guardian | **	group and object permissions, not used in our application	High
knox | auth token | **	used for authorisation tokens, not needed for our application	High
sessions | session | **	used for the sessionstorage, not needed in our application	Medium
api | medicine | can add medicine	used for adding medicines, also used for the medicine endpoint of the scraper	Medium
api | procedure | can add procedure	used for adding procedures, also used for the procedure endpoint of the scraper	Medium
api | yy | Can view xx in yy	Permission to view the variable xx from table yy	Low
api | yy | Can add yy	Add information to table yy, not used in our application	Medium

api | yy | can change yy	Edit information to table yy, not used in our application	Medium
api | yy | can delete yy	Delete information from table yy, not used in our application	Medium
api | yy | can view yy	View information from table yy, not used in our application	Low

Model documentation

The model primarily consists of classes generated by converting a MySQL Schema to a Django model using manage.py's inspectdb function. Smaller additions are made directly in Django. As both of these processes are covered extensively in their respective framework's documentation, they are omitted here for brevity. The rest consists of default classes from the Django and Knox frameworks, which remain unaltered, and the model file for the save selection table. These are likewise of limited relevance to the medicine regulation model.

The current model is represented below. All attributes are based on the variables_regscidb_20220401 with the old.xlsx file that we received.

Main tables

Medicine	Authorisation	Procedure
EUNumber EMANumber ATCCode(FK) ActiveSubstance(FK) NewActiveSubstance LegalBasis(FK) LegalScope(FK) ATMP MedicineType(FK) Status(FK) Referral Suspension EMAURL ECURL ManuallyUpdated	EUNumber(FK) Rapporteur(FK) CoRapporteur(FK) AcceleratedGranted AcceleratedMaintained AuthorisationTotalTime AuthorisationActiveTime AuthorisationStoppedTime DecisionTime DecisionURL AnnexURL EPARUrl ManuallyUpdated	EUNumber(FK) ProcedureCount CommisionNumber EMANumber ProcedureDate ProcedureType(FK) DecisionDate DecisionNumber DecisionURL AnnexURL ManuallyUpdated
Medicine and procedure tables have boolean tracking if they were manually updated by the scraper		

Supporting tables

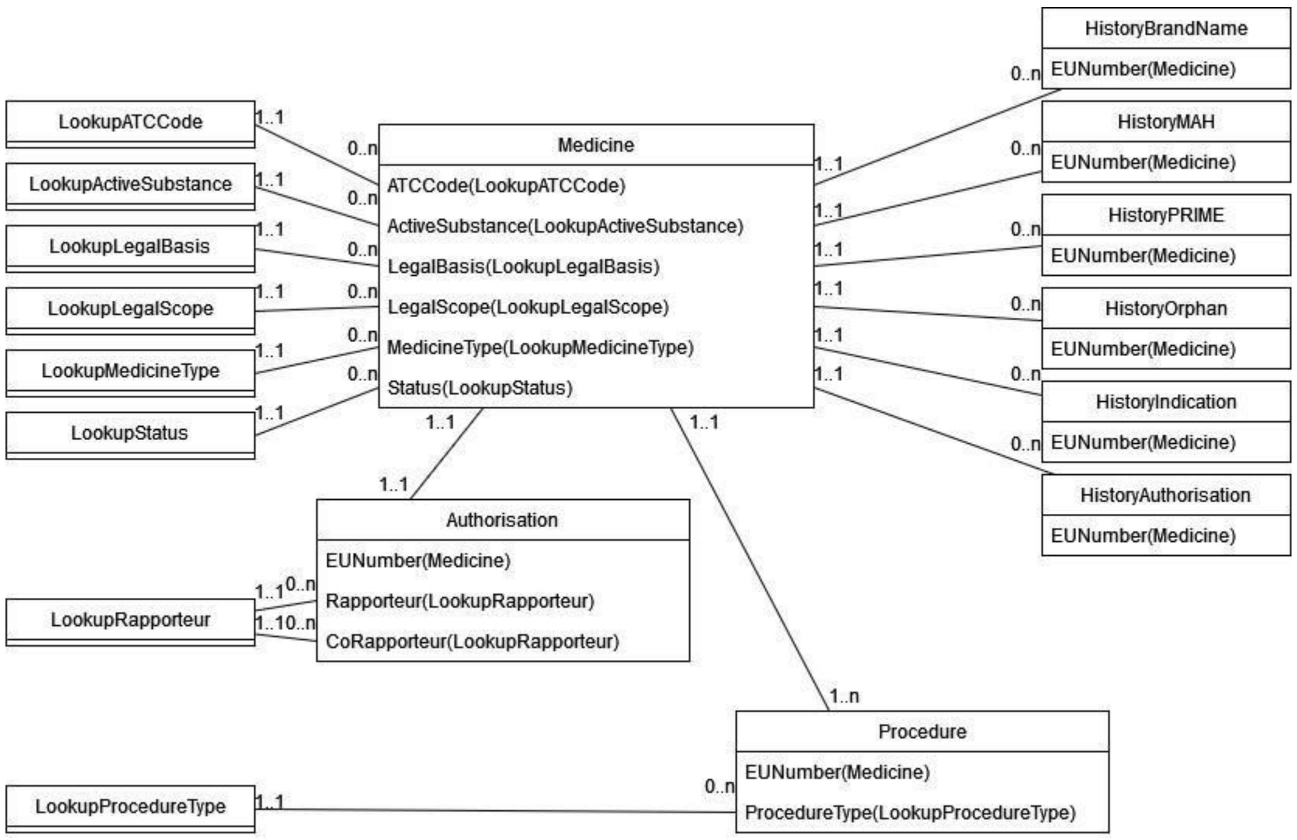
HistoryBrandName	HistoryMAH	HistoryPRIME	HistoryOrphan	HistoryIndication	HistoryAuthorisation
Medicine	Medicine	Medicine	Medicine	Medicine	Authorisation
ID EUNumber(FK) BrandName BrandNameDate	ID EUNumber(FK) MAH MAHDate	ID EUNumber(FK) Prime PrimeDate	ID EUNumber(FK) Orphan OrphanDate	ID EUNumber(FK) Indication IndicationDate	ID EUNumber(FK) OpinionDate AuthorisationDate DecisionAuthorisationType AnnexAuthorisationType RegisterAuthorisationType

Lookup Tables

LookupATCCode	LookupActiveSubstance	LookupLegalBasis	LookupLegalScope	LookupMedicineType	LookupStatus	LookupRapporteur	LookupProcedureType
Medicine	Medicine	Medicine	Medicine	Medicine	Medicine	Authorisation	Procedure
All lookup tables only have a single value identical to their name without the Lookup~ prefix							

The main tables store the information provided in the other spreadsheets. At the moment, there is a 1:1 relation between medicines and authorisations. They are in separate tables so it’s possible to change this in the future.
The history tables record changes of the specific attribute over time and are filled by the scraper.
The first four(brand name, MAH, PRIME and Orphan) are filled in at the time of this writing.
Lastly there are lookup tables for attributes that should be generally immutable like chemical formulae and the laws at the time of an authorisation.

Schematic view showing only foreign keys and relations:


HistoryBrandName
EUNumber(Medicine)
HistoryMAH
EUNumber(Medicine)
HistoryPRIME
EUNumber(Medicine)
HistoryOrphan
EUNumber(Medicine)
HistoryIndication
EUNumber(Medicine)
HistoryAuthorisation
EUNumber(Medicine)
LookupATCCode
LookupActiveSubstance
LookupLegalBasis
LookupLegalScope
LookupMedicineType
LookupStatus
Medicine
ATCCode(LookupATCCode)
ActiveSubstance(LookupActiveSubstance)
LegalBasis(LookupLegalBasis)
LegalScope(LookupLegalScope)
MedicineType(LookupMedicineType)
Status(LookupStatus)
Authorisation
EUNumber(Medicine)
Rapporteur(LookupRapporteur)
CoRapporteur(LookupRapporteur)
LookupRapporteur
Procedure
EUNumber(Medicine)
ProcedureType(LookupProcedureType)
LookupProcedureType
1..1
0..n
1..n